TAMPERE UNIVERSITY OF TECHNOLOGY

SAMPO SUONSYRJÄ

TOWARDS NON-INVASIVE COLLECTING OF SOFTWARE ANALYTICS DATA

Master of Science Thesis

Prof. Samuli Pekkola and Prof. Tommi Mikkonen have been appointed as the examiners at the Council Meeting of the Faculty of Business and Technology Management on October 8, 2014.

ABSTRACT

SUONSYRJÄ, SAMPO: Towards non-invasive collecting of software analytics data
Tampere University of Technology
Master of Science Thesis, 70 pages, 2 appendices (4 pages)
February 2015
Master's Degree Programme in Information and Knowledge Management
Major: Business Information Management
Examiners: Professor Samuli Pekkola, Professor Tommi Mikkonen

Keywords: Aspect-Oriented Programming, analytics, software development, software operation data, data collecting, data mining

The development of new methods and technologies in software engineering has radically changed not only software itself but also the way software is developed. These new approaches, such as Agile practices, promise to produce software, which suits its users better than before by increasing communication with users.

Software analytics has this same goal of making better software, but by increasing the amount of real usage data in decision making. However, traditional methods for collecting data regarding how users use software systems cannot usually produce that data in such quantities that are required by new methods such as A/B testing. In this sense, interviews, observation, and questionnaires can gather valuable qualitative data, but more sophisticated new methods are needed to assist in collecting especially quantitative software operation data.

In this thesis, the use of Aspect-Oriented Programming was studied for gathering software operation data. Design science was used as the research strategy, and as its outcome a tool for collecting software operation data was developed using AspectJ language. For evaluation, this tool was used to collect data from a demo application of the Vaadin Framework.

Aspect-orientation allowed the tool to be developed in isolation from the demo application. This way, no alterations to the demo application's source code were needed, which again allowed its evolution without compromising its compatibility with the collecting tool. The Vaadin Framework provided the collecting tool with abundant contextual data, which can be used for evaluating user behaviors. The developed tool can be reused with other Vaadin Framework applications, and it provides Vaadin a starting point for creating an additional analytics feature for their framework.

TIIVISTELMÄ

SUONSYRJÄ, SAMPO: Ohjelmistojen analytiikkadatan huomaamaton kerääminen
Tampereen teknillinen yliopisto
Diplomityö, 70 sivua, 2 liitettä (4 sivua)
Helmikuu 2015
Tietojohtamisen diplomi-insinöörin tutkinto-ohjelma
Pääaine: Tiedonhallinta
Tarkastajat: professori Samuli Pekkola, professori Tommi Mikkonen

Avainsanat: Aspektiohjelmointi, analytiikka, ohjelmistotuotanto, ohjelmistojen käyttödata, tiedon kerääminen, tiedonlouhinta

Ohjelmistotuotannossa käytettyjen menetelmien ja teknologioiden kehittyminen on johtanut paitsi uudenlaisiin ohjelmistoihin myös kokonaan erilaisiin tapoihin kehittää ohjelmistoja. Näiden uudenlaisten suuntausten – kuten ketterien menetelmien – yhtenä lähtökohtana on parantaa kommunikaatiota käyttäjien ja kehittäjien välillä, minkä tulisi johtaa käyttäjille paremmin sopiviin ohjelmistoihin.

Ohjelmistoanalytiikka pyrkii vastaavasti tekemään ohjelmistotuotannon päätöksenteosta datalähtöisempää keräämällä tietoja siitä, miten aidot käyttäjät käyttävät ohjelmistoja, ja analysoimalla näitä tietoja jatkokehitystä varten. Perinteiset tavat kerätä tällaisia tietoja eivät hitautensa takia vastaa enää kaikkiin nykypäivän vaatimuksiin sillä esimerkiksi A/B-testaamisen toteuttamisessa suuri määrä kvantitatiivista tietoa on arvossaan toisella tavalla kuin mitä on kerättävissä esimerkiksi observoimalla ohjelmiston käyttöä. Uusia menetelmiä myös käyttödatan keräämiseen onkin siis etsittävä jatkuvasti.

Tässä työssä tutkittiin aspektiohjelmoinnin käyttöä ohjelmistojen käyttödatan keräämisessä. Tutkimusstrategiaksi valittiin design science, jossa tieteellisten kirjallisuuskatsausten pohjalta kehitettiin AspectJ-kielellä työkalu ohjelmistojen käyttödatan keräämiseen. Tämä keräystyökalu liitettiin Vaadin-sovelluskehyksellä kehitettyyn demo-sovellukseen, minkä yhteydessä keräystyökalun ja ylipäätään aspektiohjelmoinnin mahdollisuuksia arvioitiin käyttödatan keräämisessä.

Aspektiohjelmointi mahdollisti keräystyökalun kehittämisen erillään demo-sovelluksesta, jonka lähdekoodiin ei jouduttu tekemään muutoksia. Tämä taas mahdollisti demo-sovelluksen kehittymisen ilman kommunikaatiota keräystyökalun kehittäjään ja silti yhteensopivuuden jatkumisen demo-sovelluksen ja keräystyökalun välillä. Vaadin-sovelluskehys tarjosi ylipäätään paljon mahdollisuuksia käyttötietojen keräämiseen, mitä voidaan hyödyntää arvioitaessa sillä kehitettyjen sovellusten käyttäjien käyttäytymistä. Keräystyökalua voidaan käyttää uudelleen muiden Vaadin-sovelluskehyksellä kehitettyjen sovellusten kanssa, ja se tarjoaa samalla myös yhden mahdollisen lähtökohdan kehittää analytiikkaominaisuuksia ylipäätään Vaadin-sovelluskehykseen.

PREFACE

This thesis is a result of Digile's Need for Speed program. It has been an exciting privilege to get to know some of the finest software-intensive companies in Finland, and I cannot overemphasize my gratitude to Digile for this opportunity. One of these companies is Vaadin, whose passionate COO Jurka Rahikkala showed a special interest on the topic from the very beginning of the thesis project. I would like to thank him for his fast-paced feedback on the thesis and overall for the opportunity to do research on the Vaadin framework.

I am grateful for having met a countless number of friends, co-workers, lecturers, and instructors during my time in different schools and especially at Tampere University of Technology. I look up to you and thank you for getting me to this point in life. Of these, I would like to extend my gratitude to my opponents in the Master's Thesis seminar, N4S project team members, and especially Professor Tommi Mikkonen of the Department of Pervasive Computing. He trusted me with the topic from the very beginning and along the way he gave me some expert advice and guidance as needed. Likewise, I thank Professor Samuli Pekkola of the Department of Information Management and Logistics for sharing his expertise with me and always making me exceed myself.

Finally, I want to take the opportunity to thank my wonderful family and especially Krista for all of their unconditional and unending support, trust, and love.

Tampere, January 15, 2015

Sampo Suonsyrjä

TABLE OF CONTENTS

ABBREVIATIONS

AJAX	Asynchronous JavaScript and XML
API	Application Programming Interface
AOP	Aspect-Oriented Programming
COTS	Commercial Off-The-Shelf
CSS	Cascading Style Sheets
GWT	Google Web Toolkit
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IDE	Integrated Development Environment
IP	Internet Protocol
IS	Information Systems
IT	Information Technology
MVP	Minimum Viable Product
NITCAD	Non-Invasive Tool for Collecting Analytics Data
OOP	Object-Oriented Programming
SQL	Structured Query Language
SCSS	Sassy CSS
SI	Software Intelligence
SOK	Software Operation Knowledge
UI	User Interface
URI	Uniform Resource Identifier
XML	Extensible Markup Language

1. INTRODUCTION

1.1. Background and motivation

The development of new methods and technologies in software engineering has radically changed not only software itself but also the way software is developed (Fugetta & Di Nitto 2014). These new approaches, such as Agile practices, promise to produce software, which suits its users better than before by increasing communication with users (Fowler & Highsmith 2001). Software analytics has this same goal of making better software, but by increasing the amount of real usage data in decision making.

This decision making in software development consists of determining things such as when a software system is ready for release, whether a part of a software system should be re-factored or re-written, or which parts of a software system should be thoroughly tested (Hassan & Xie 2010, p.161). As software development is a data-rich activity (Buse & Zimmermann 2011, p.1) these decisions would not have to be based on a gut-feeling but rather on cold hard facts. Analytics offer help in this and as such, Buse & Zimmermann (2011, p.3) state the goal of analytics as "…to help managers move beyond information and toward insight".

Overall, analytics is an expansive field with a large collection of different techniques and methods and with several implementation contexts. According to Chen et al. (2012, p.1166) analytics "…is often referred to as the techniques, technologies, systems, practices, methodologies, and applications that analyze critical business data to help an enterprise better understand its business and market and make timely business decisions." They continue with stating how these can be applied to several different industries such as e-commerce, market intelligence, e-government, healthcare, and security. (Chen et al. 2012, p.1166) Similarly, Begel & Zimmermann (2014, p. 1) claim that during the recent years also software engineering researchers have understood the opportunity to use more data for constantly making better decisions. However, as even sporting teams have improved their performance with the help of analytics, the uses for analytics seem to be fairly general and broad. (Begel & Zimmermann 2014, p.1)

The field of software analytics gains its empowering data from various sources such as source code repositories, bug reports, user-experience surveys and documentations. The focus of this research, however, lies primarily on software operation knowledge. Kristjansson & van der Schuur (2009) have formulated the concept as follows. They describe that to consist of knowledge of in-the-field performance, quality and usage of software, and knowledge of end-user experience and end-user feedback. The researchers

continue with stating how software vendors have a great interest in acquiring such knowledge, but that "the systematic practice of gathering, analyzing and acting on such knowledge is still limited." (Kristjansson & van der Schuur 2009)

Tools such as Google Analytics have been gaining popularity recently in the field of web analytics. The tool enables developers to track information regarding the users such as their geographical location, time spent on site, and the path they took to get there (Google, Inc. 2014). As crucial as this information is for software development, Google's framework operates primarily in the Web, and therefore it does not help out with generic desktop application development.

Back in the day with the Web still not enabling the collecting of such quantities of data, the usage of software has been studied anyway. Of course, the methods have been somewhat different. For example, Holzinger (2005) has listed approaches to collect data regarding software usability. These include thinking aloud, field observation, and questionnaires. Johnson (2013) continues the list for software process data with describing how developers have for example had to fill out 12 separate forms including time and defect recording logs, code and design checklists, and so on. Obviously, this requires vast amounts of manual work and practically makes it impossible to use the methods continuously.

In order to fulfill the ever more complex needs of today – such as demands for real-time analytics – more sophisticated tools to collect software operation data are needed. Aspect-Oriented Programming (AOP) might present facilities to gather software operation data in a more advanced manner. This hopefully means both easier data collecting and larger amounts of software operation data as well as whole new types of data.

The defining idea of AOP is in separation of crosscutting concerns. Concerns are behaviors in a computer program, and if their implementation is scattered across many modules of the program they are said to be crosscutting. (Asif & Reddy 2013) They are said to range from high-level notions such as security and quality of service to low-level notions like caching and buffering (Elrad & Filman 2001). Without AOP, implementations of crosscutting concerns leave the source code tangled and scattered, and all in all difficult to maintain (Kickzales et al. 1997). To overcome this, the promise of AOP is to enable developers to cleanly separate these crosscutting concerns, and decrease the level of code scattering and tangling (Subotic & Bishop 2005).

In order to accomplish this, AOP increases the level of modularization. Filman et al. (2004) have formulated the way aspects work as follows:

In programs P, whenever condition C arises, perform action A.

In Figure 1 this pattern is recurring with the execution flow of program P on the left, several different conditions C arising along the way, and corresponding actions A on the right performed by aspects. In this case, aspects are used for creating logging for the program. This example represents neatly one of the benefits of AOP, as the additional code performing the logging is defined clearly as its own module and no insertions to the original source code are needed.

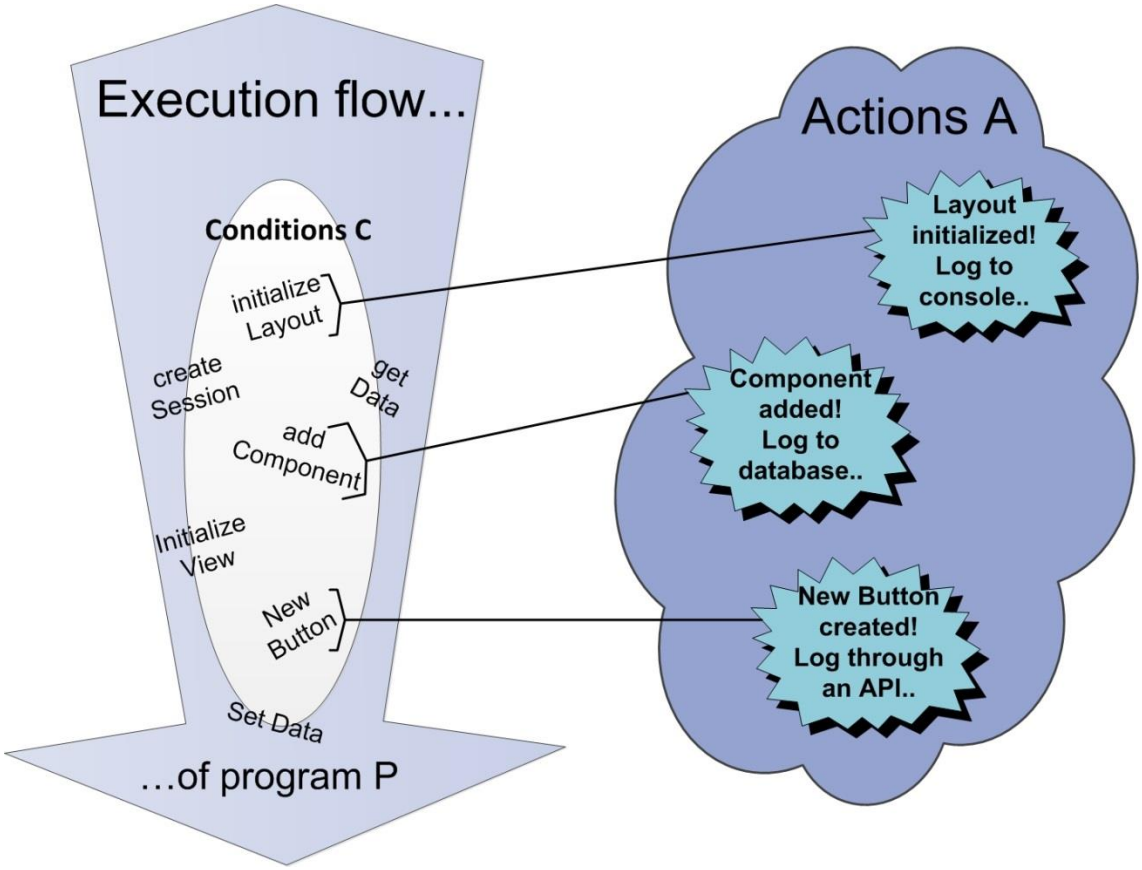

Figure 1 *Aspects attaching to points in a program's execution*

According to Kumar et al. (2009) AOP has gained popularity with adoptions in custom programs and some frameworks such as the Spring framework. They also list AOP's benefits as increasing the scope of concerns that can be captured cleanly, having explicit language support, and providing an elegant mechanism of separation for custom solutions. Classical areas for AOP have been logging, error handling, and audit events. (Kumar et al. 2009) Asif & Reddy (2013) extend this list with access control, performance, tracing, and with studying AOP for software testing and debugging. In addition, Chaudry & Chatterjee (2013) have done research on AOP and its relation to the reusability of software. Extending these uses, this research aims to study AOP with analytics.

1.2. Research problems

This research is focused on studying the concepts of AOP and software analytics. In particular, the research problem is to find out:

- *How can AOP be used for supporting software analytics?*

To answer this main research question, the following sub-questions can be derived.

Firstly, this research is set to study the basics of AOP. As a programming paradigm, it is obvious that the field of research with AOP is extensive. Therefore, the focus of this research in the field of AOP is to find out the very basic concepts of the paradigm and to study how one can implement ideas with an AOP language. The following sub-question formulates the focus of this research in the field of AOP:

1. *What kind of advantages AOP offers for collecting software operation data?*

Secondly, the concept of software analytics is studied, in which the focus is mainly on software operation knowledge. This sets the next sub-question for the research as follows:

2. *What kind of software operation knowledge is needed for software analytics?*

The first sub-question is set to assist in the technical aspects of understanding how to collect software operation data with AOP and the second one in comprising a relevant framework for analytics in software development. The third and most ambitious research goal is to combine the first two research goals and develop a Non-Invasive Tool for Collecting Analytics Data (NITCAD) to collect software operation data in an AOP language and implement it in a suitable context. The third and final sub-question to guide the research is formulated as follows.

3. *How NITCAD is able*
   3.1. to deliver the benefits AOP paradigm promises and
   3.2. to answer to the information needs of software analytics?

To accomplish the goals set in the third sub-question, a target application framework is selected. On one hand the objectives of this research are academic, but at the same time there are possibilities to take commercial advantage of the results. So, on the other hand the target application framework is selected not only for the academic fit but also for the benefit of the Digile's Need for Speed program and its objectives (Huomo et al. 2013).

If the results of this research signal that AOP is a sensible way to collect software operation data, Vaadin – the case company of this research – is going to possess a valuable starting point for creating new business opportunities in software analytics. In

any case, the conclusions of the research are going to contribute to the academic research on ways to utilize AOP and implement software analytics altogether.

1.3. Philosophy and approaches of the research

Although there are no right answers to questions such as "what to study, what kind of material should be collected, or which method would suit the best", the selecting between one and the other is quite important (Hirsjärvi et al. 2007, p.119). Still, all of these questions can be valued secondary when compared to the underlying world view or philosophy of the researcher (Saunders et al. 2009, p.106). As this philosophy is either a conscious or an unconscious choice for the researcher (Hirsjärvi et al. 2007, p.119), respectively the resulting selections concerning for example research design are based on either conscious or unconscious choices. As implied in the previous sentence, the choices of research design can be seen to result from the selection of a research philosophy (Saunders et al. 2009, p.108). Saunders et al. (2009, p.108) present the flow of these choices in form of a research "onion", which is portrayed in Figure 2.

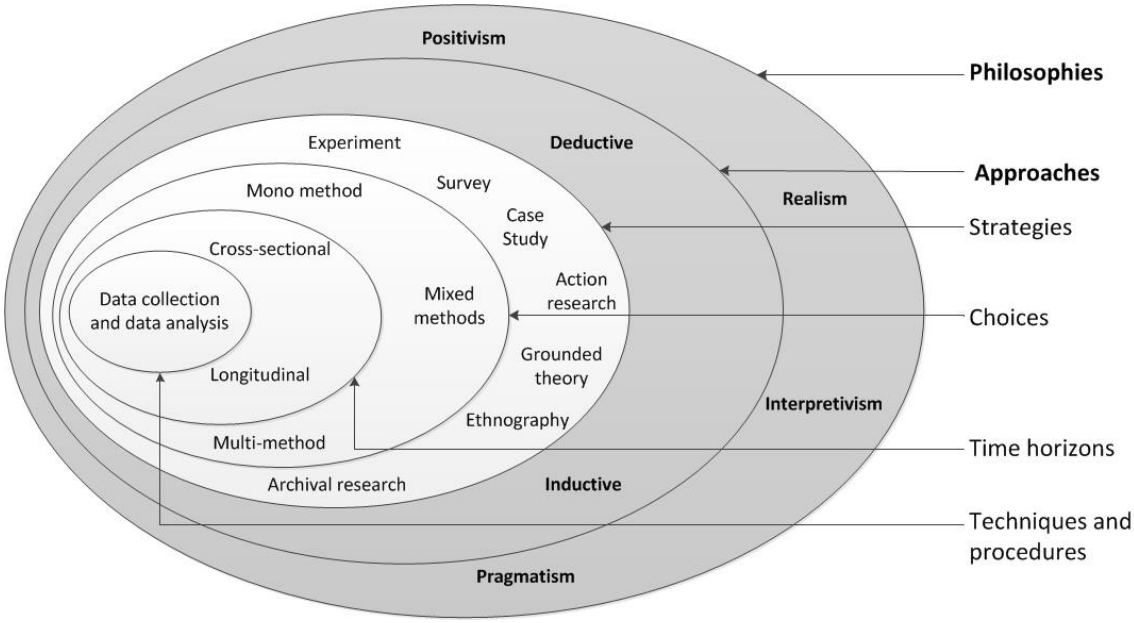

Figure 2 *Research "onion" (adapted from Saunders et al. 2009, p.108)*

What comes to research philosophies, there seems to be several ways of separating and categorizing them. Olkkonen (1994, p.26) distinguishes positivism and hermeneutics as the prevailing philosophies, but in addition to positivism, Saunders et al. (2009, p.108) introduce also realism, interpretivism, and pragmatism, leaving out hermeneutics. However, they argue that the researcher should see the different philosophies more as a continuum than distinct positions. Despite the popularity of the philosophies in a certain research field or time, they can be studied from the basic philosophical points of view of ontology and epistemology.

If the research philosophies are organized in the form of a continuum as proposed above, positivism would be on one end of the line. The research with an underlying positivist philosophy usually resembles the research a natural scientist conducts. Only observable data is considered to lead to credible results and the aim is to gain law-like generalizations. (Saunders et al. 2009, p.113-114) Moving on, realism can be considered a step away from positivism and Olkkonen (1994, p.27) presents it to emphasize the external observable reality independent of the human mind. It is also seen more as a general discipline of philosophy by Olkkonen. Saunders et al. (2009, p.119) describe positivism and realism to be quite similar ontologically, and even the epistemological differences seem to be only on the foci of positivism generating law-like generalizations and realism explaining things in their own contexts.

Interpretivism, on the other hand, stands out of the aforementioned philosophies as its ontological view is a more subjective one. Saunders et al. (2009, p.116) describe interpretivism to be a valid philosophy for example for business research, where the researcher has to adapt to an empathetic role to better understand the subjects of the research. The strength of this perspective is that with that it is possible to understand the complex and unique situations among an organization and the social actors it consists of. Ontologically, interpretivism recognizes multiple realities and the epistemological focus is on the details of a situation from the different viewpoints of its social actors. (Saunders et al. 2009, pp.115-119)

Hermeneutics and pragmatism can be seen in the other end of the continuum. Olkkonen (1994, p.27) describes hermeneutics to emphasize the understanding of phenomena, and Saunders et al. (2009, p.119) describe similarly pragmatism to aim at interpreting data. Unlike interpretivism though, pragmatism is seen to gain data from either or both observable phenomena and subjective meanings. Ontologically, this means that pragmatism identifies multiple realities and the view is actually chosen to best answer the research questions. Therefore, there can be multiple ontological and epistemological views in one research and the researcher can change between them respecting the given research question. (Saunders et al. 2009, pp.109&119)

Back in the philosophical options of this research, the choice leans quite naturally towards a pragmatist view. The first two of the research questions could outline the research towards a philosophy of realism as their aim is to gain a broad look at the given subjects. That could be seen almost as an aim for a law-like generalization for example for the research question #1 (*What kind of advantages AOP offers for collecting software operation data?*). Then again, answering the question #3 with a more subjective mindset seems justified. Given these kind of varieties in the goals of the research questions, the multiple ontological and epistemological possibilities of a pragmatic research could work best for this research.

Correspondingly to the varieties in the ontological and epistemological mindsets of this research and the pragmatic view, the research approaches used are multiple as well. The first two questions are approached with deduction. Basics of AOP and software analytics are first found out with literature reviews, after which hypotheses are made and tested with the specific case of this research. Then again, an inductive approach is used with answering the third question. As this question is about the specific and single case, there is no need to achieve generalization but rather an understanding.

1.4. Study outline

To respond to the set research questions with the selected research approach, the rest of the thesis is structured as follows.

Firstly, Chapter 2 describes the different research strategies, methods, techniques and other details concerning the way this research is conducted. The grounds for selecting the particular strategies are introduced and the practicalities of the research execution are described in detail.

Secondly, literature reviews on AOP and software analytics are presented in Chapters 3 and 4 respectively. These cover the deductive part of the study and respond to the first two of the research questions.

Thirdly, the structure of this thesis is continued with answering to the third research question. Chapter 5 explains the background of the case by describing the case company and its web application framework. This is followed by Chapter 6 describing the results of the study. These include the tool for collecting data as well as the types of data. These chapters cover the empirical part of the study.

Finally, the study and its results are evaluated in Chapter 7, and final conclusions are drawn in Chapter 8.

2. RESEARCH DESIGN

This chapter describes the different research strategies, methods, techniques and other details concerning the way this research is conducted. The grounds for selecting the particular strategies are introduced and the practicalities of the research execution are described in detail.

2.1. Research strategies

One of the primary objectives of this research is to study the attributes of AOP in the use for software analytics. This being a relatively novel research topic, the number of available frameworks or tools for the job was very limited. In addition, Digile's Need for Speed (N4S) program (http://www.n4s.fi/fi/) encourages its research efforts to produce results not only for academia but also for the project partners of practice. Thus, the objectives of this research are twofold. Firstly, the aim of the study is to make a scientific contribution and secondly, the results of this research should also assist practitioners in solving the anticipated needs of applying AOP to collecting software operation data. Fortunately, this kind of a dual mission is generally a preferred one for Information Systems (IS) research (Sein et al. 2011).

Cole et al. (2005) claim that there are two research strategies in the IS field, which can execute this kind of two-way ambition. These are design science research (see Hevner et al. 2004; Peffers et al. 2007) and action research (see Baskerville & Meyers 2004; Saunders et al. 2009). Despite some differences in the aforementioned strategies, there have been organized efforts for comparing and even combining these (see Cole et al. 2005; Sein et al. 2011). To make the appropriate choice of research strategy, both of the aforementioned strategies are studied and presented as follows below. Afterwards, the preferred research strategy is selected and followed throughout the rest of the research.

When considering the philosophical aspects of this research, both design science and action research fit in well with pragmatism. On one hand, there are multiple states of the world, or the tool, as it is being developed and looked at from different subjective perspectives. This contrasts the objective views of positivism and realism, where a single state reality is studied. On the other hand, the underlying physical reality is seen as a single state reality, which again contrasts the interpretivistic philosophy. However, the chosen pragmatic research philosophy gives the necessary room to work within the boundaries of these views and emphasize the utility of the tool being developed.

Action research was introduced already in 1946 by Kurt Lewin and nowadays it is considered a well-established research strategy (Cole et al. 2005, p.4). Saunders et al. (2009, p.147) present action research more as a strategy for social research, and specifically for organizational change. However, their description of its process (see Figure 3) consisting of diagnosing, planning, taking action and evaluating (Saunders et al. 2009, p.147) seems to fit in well within IS research too. Cole et al. (2005, p.4) continue by stating how the goal of action research is on one hand a resolution to a practical problem and on the other a scientific contribution. This is naturally nicely in line with the goals of this research. In addition, Saunders et al. (2009, p.147) point out that this scientific contribution does not necessarily have to be forming of a scientific theory but transferring of knowledge and using it in another context (i.e. consultancy) goes as well.

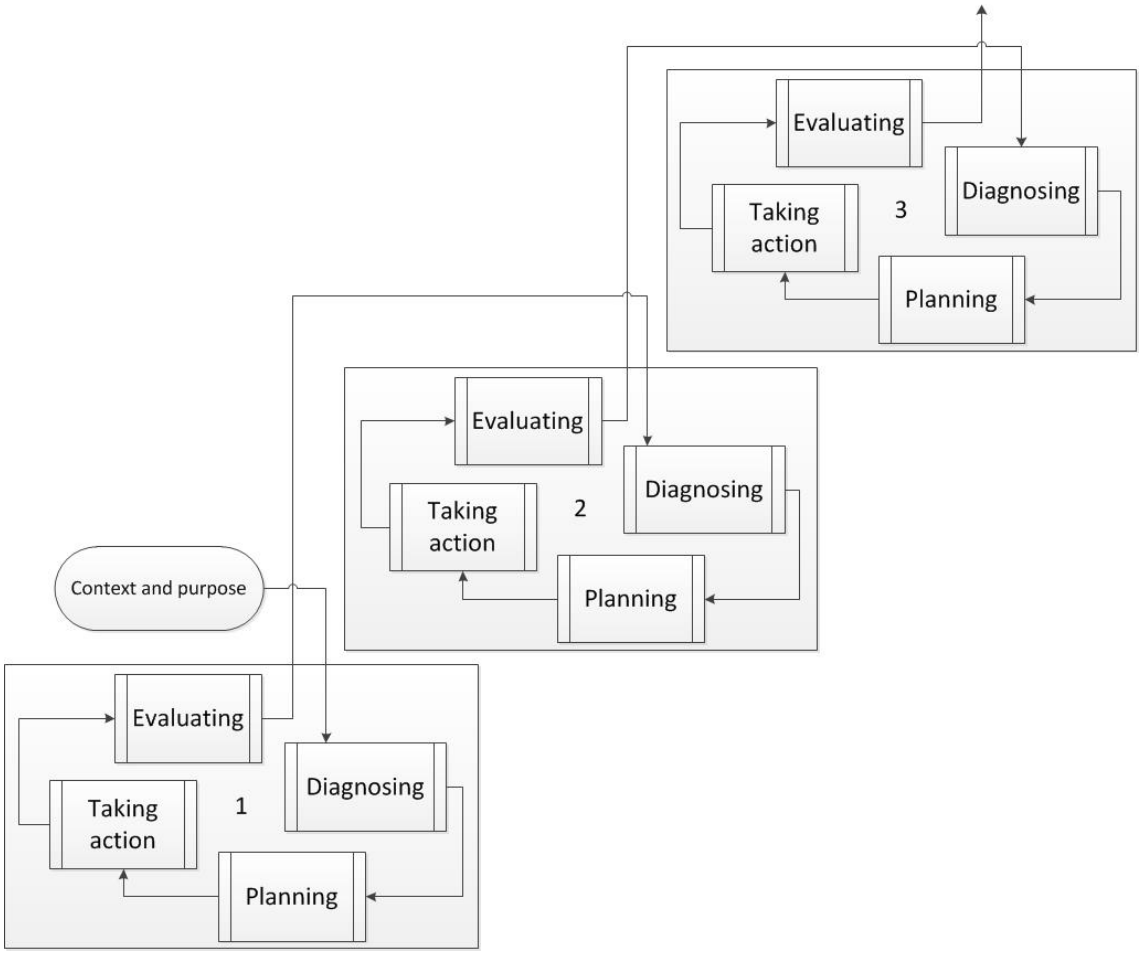

Figure 3 *Action research process (adapted from Saunders et al. 2009, p.148)*

However, there are other research strategies that should be evaluated for the purposes of this research in addition to action research. As mentioned one of them is design science, which shares many similarities with the process of action research. These similarities can be seen by comparing Figure 3 and Figure 4, of which the latter illustrates the research process of design science.

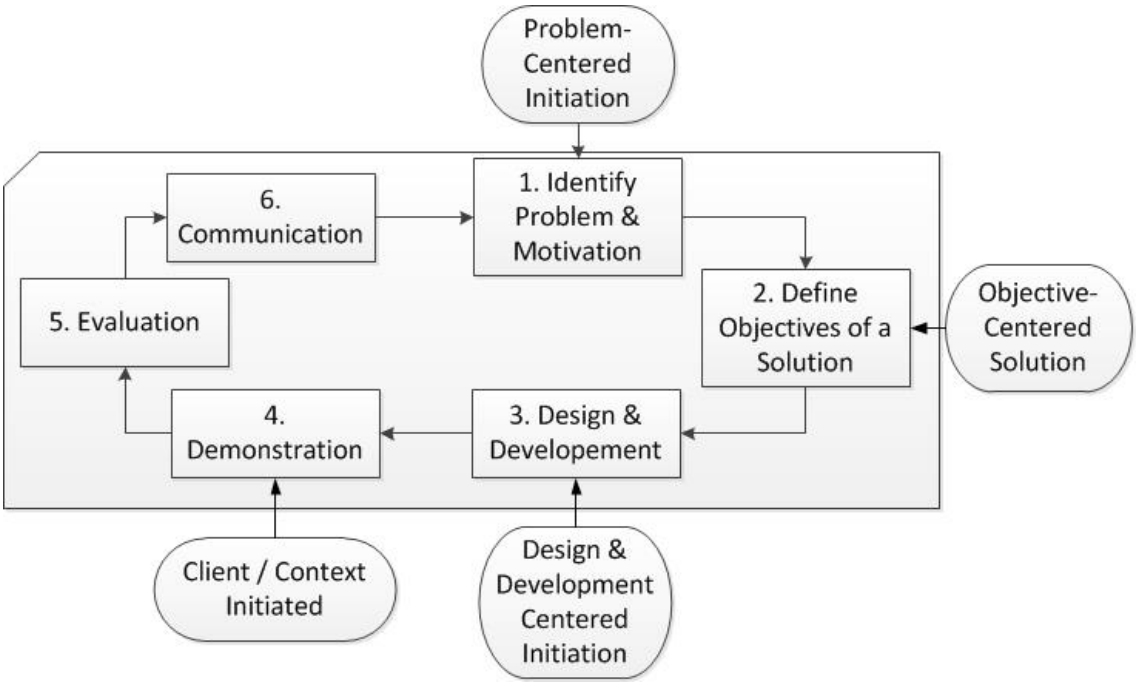

Figure 4 *Process model of design science (adapted from Peffers et al. 2007, p.54)*

Both of the processes are iterative and they are constructed around the idea of identifying specific problems in a context, designing resolutions to answer them, and evaluating these answers in that context. However, in contrast to action research Peffers et al. (2007, p.54) present design science process to have multiple possible entry points for the research as seen in Figure 4. Despite this specific difference, which is more of a practical kind, a more substantial dissimilarity can be recognized in the focus of design science, which is more technology oriented from the start. This can be seen for example in Peffers et al. (2007, p.49) presenting design science as creating and evaluating information technology (IT) artifacts, which are constructed to solve organizational problems.

Similarly to action research, design science aims at balancing on the two-folded demands of science and practice. As seen in Figure 4 the research process can even start as "client initiated". However, Peffers et al. (2007, p.54) have defined design science to have other possible entry points as well. These are problem, objective, and design & development centered initiations. Considering these multiple entry points, it seems that design science is formed to ease the starting of the process regardless of the stand point of the initiator, and to output regular evenly balanced results for both academia and practice. Correspondingly, Cole et al. (2005, p.3-4) support this view by pointing out four possible outputs for the process: constructs, models, methods, and instantiations.

The nature of these outputs highlights also the difference between design science and action research. In contrast to behavioral IS research aiming at "truth", design science targets "utility". This means that while behavioral IS research is looking for exploration and validation of generic cause-effect relations design science research constructs and

evaluates generic means-ends relations. (Winter 2008, p.470) This view is shared already by March & Smith (1995, p.253) as they stress the difference in natural science trying to understand reality and design science attempting to create things to serve human purposes.

All in all, both the action research and the design science strategies are seen philosophically fit for this research. However, the practical aspect of design science brings it naturally closer to the research goal of developing an IT artifact. Therefore, the design science strategy is selected and followed throughout the research to ensure the scientific rigor of this research. As mentioned above, there have been also efforts to combine design science and action research, but as Sein et al. (2011, p.39) point out, their intention with this combination is to increase the organizational interventions of the process. However, this will not be necessary considering the circumstances of this research, and thus the plain design science strategy is quite adequate enough to move on with. Figure 5 illustrates the selected path for this research with the aforementioned research "onion" by Saunders et al. (2009, p.108).

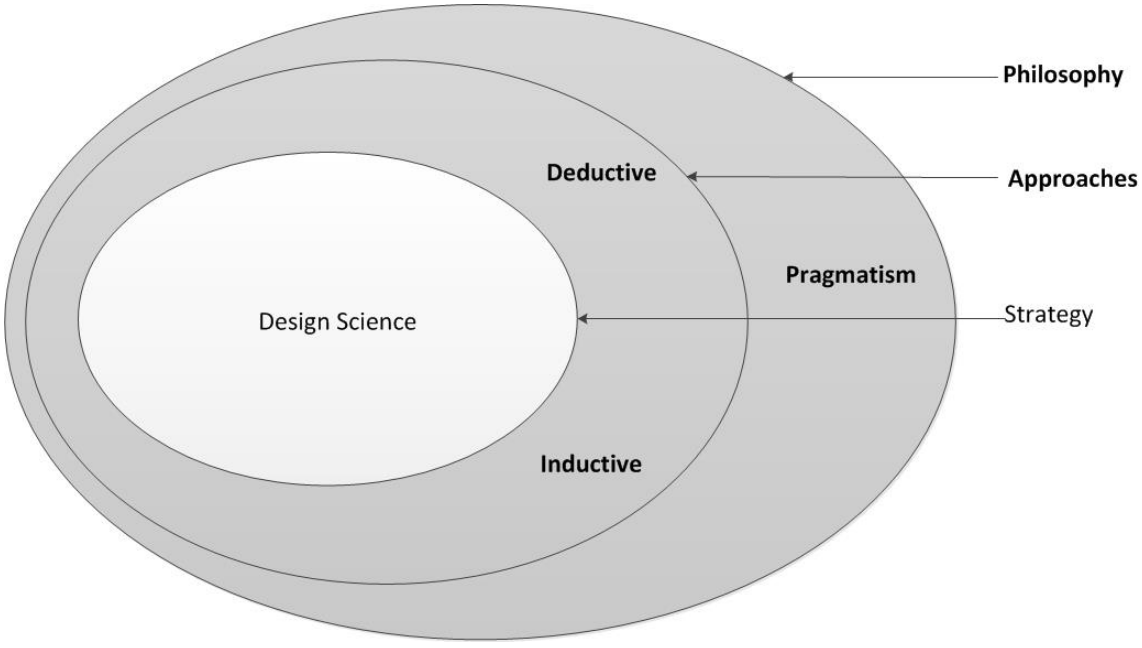

Figure 5 *The flow of research design related decisions in this study*

The flow of research related decisions are combined in Figure 5 starting from the selected research philosophy and leading to the research strategy of design science. Hevner et al. (2004) have produced seven guidelines for design science in information systems research and they, along with their implementation for this research, are presented next.

2.2. Research methods

Peffers et al. (2007) raise the concern that there has been a lack of a commonly shared design science research process model in IS. They see this resulting for example in IS researchers publishing their work in engineering journals, where design science methods are more common (Peffers et al. 2007). Therefore, there are multiple reasons for this research to follow the practical guidelines presented by Hevner et al. (2004).

2.2.1. Guidelines for research

Firstly, if this research is looked at as a piece of scientific research, it is intended to support the reasoned use of rigorous scientific methods in IS research. This, again, is to support the humble aspiration of the researcher to increase the appreciation of the IS as a research field. Secondly, the intention is to underpin the generally accepted mental model of design science within the field of IS. This kind of mental model was also an intention of Peffers et al. (2007) in their effort of formalizing design science. Thirdly, the application of rigorous research methods is one of the guidelines for design science presented in Hevner et al. (2004). Finally, as this research is also a thesis, it aims at clarifying how the researcher has learned to look for fitting research strategies, select a proper one, and document its use. In order to reach these aims, the guidelines are summarized in Table 1 and described in more detail as follows.

Table 1 *Research guidelines for design science. Adapted from Hevner et al. (2004 p.83)*

#	**Guideline**	**Description**
1	Design as an Artifact	Design-science research must produce a viable artifact in the form of a construct, a model, a method, or an instantiation.
2	Problem Relevance	The objective of design-science research is to develop technology-based solutions to important and relevant business problems.
3	Design Evaluation	The utility, quality, and efficacy of a design artifact must be rigorously demonstrated via well-executed evaluation methods.
4	Research Contributions	Effective design-science research must provide clear and verifiable contributions in the areas of the design artifact, design foundations, and/or design methodologies.
5	Research Rigor	Design-science research relies upon the application of rigorous methods in both the construction and evaluation of the design artifact.
6	Design as a Search Process	The search for an effective artifact requires utilizing available means to reach desired ends while satisfying laws in the problem environment.
7	Communication of Research	Design-science research must be presented effectively both to technology-oriented as well as management-oriented audiences.

Hevner et al. (2004, p.83) point out how the IT artifacts constructed as the output of design science are rarely full-scale IT systems for practice use. Rather, they are more likely presentations and examinations of novel ideas and innovations. As the guideline #1 suggests though, the main purpose is to produce a viable artifact. In this case the aspect-oriented NITCAD presented in Chapter 6 fulfills the aim of this guideline. Its nature is, however, very much in line with the claim of a design science artifact being a novel proposal rather than a full-grown system.

The second guideline emphasizes the relevance of the problem being solved. In brief, automated data gathering is argued as an essential benefit in software development, and new improvements and innovations for this process are continuously needed as modern methods for developing software rely on getting user feedback faster and in larger quantities than before.

Thirdly, the guidelines steer research processes to a rigorous evaluation of their design artifacts. This can be done with focusing on for example their "…functionality, completeness, consistency, accuracy, performance, reliability, usability, fit with the organization, and other relevant quality attributes." (Hevner et al. 2004, p.85) Being inherently an iterative process, design may begin as simplified conceptualizations and evolve from there. Evaluation benefits this by providing useful feedback to the process. Additionally, there are multiple methods for the evaluation of designed artifacts. As such, Hevner et al. (2004, p.86) have categorized them as observational, analytical, experimental, testing, and descriptive methods. Of these, descriptive methods are used for the evaluation of the tool developed in this research, and these are described in more detail in Chapter 7.

The objective is to gain knowledge of at least NITCAD's functionality, completeness, and potential influence on the field of gathering software operation data. Of the descriptive evaluation methods, informed argument method is used for building convincing arguments for the tool's utility. Although Hevner et al. (2004, p.86) introduce the descriptive methods with some caution for being suitable for the evaluation of only "especially innovative artifacts for which other form of evaluation may not be feasible", the tool developed in this research fulfills this call of being novel enough. In addition, the other methods mentioned by Hevner et al. (2004, p.86) are potential ways for the evaluation in future work, but within the scope of this thesis they are likely too extensive.

The guideline #4 directs towards satisfactory research contributions. These can be the artifact itself, foundations, and/or methodologies. In this case, the developed tool (NITCAD) is similarly the natural contribution of the research, but as there might be

additional contributions in form of foundations and methodologies too, they are discussed in Section 7.1.

The rigor of the research should be applied in a sufficient relation to the applicability and generalizability of the artifact, according to the guideline #5. Hevner et al. (2004, p.88) mention that "an overemphasis on rigor can lessen relevance", but specifically in this case being a thesis, a somewhat overemphasized approach to the rigor of the research seems justified.

Considering the guideline #6, design science requires knowledge of different domains such as requirements, constraints, and technical and organizational areas. However, as the guideline emphasizes the iterative nature of the research process, the starting point can be narrower and the progress of the search process expands the scope afterwards. For this research this kind of an approach is likely to be the most appropriate as the starting point is to gain knowledge of technology (see Chapter 3), requirements (see Chapter 4), and constraints (see Chapter 5). This leaves the organizational knowledge out and for future work.

What comes to the final guideline (#7), this research is conducted in a particularly fitted context being part of the N4S program. The communication of research should be aimed on one hand at the technology oriented and on the other hand at the managerial audiences. To answer these calls, the technical aspects of the solution are discussed in detail in Chapter 6. In addition, Chapters 4 and 7 along with the motivation section above answer to the calls for problem importance and solution effectiveness.

Looking back to the background and goals of this research, the entry point is a problem-centered initiation although the needs of the case company have had a substantial effect on the initiation too. This being the case, the order of the process points illustrated in Figure 4 is similar to how this research is conducted. To understand more profoundly how this process is implemented in the case of this research, the process is described in more detail as follows.

2.2.2. Research process

To present the research process as it is executed Figure 6 illustrates the order of the components of design science strategy and the specific corresponding methods used in this research. The center piece in the illustration is the outcome artifact of the process, and around that, inside the rectangle box, are the standard parts of the design science process. Attached to these are, however, the specific subjects concerning this specific research.

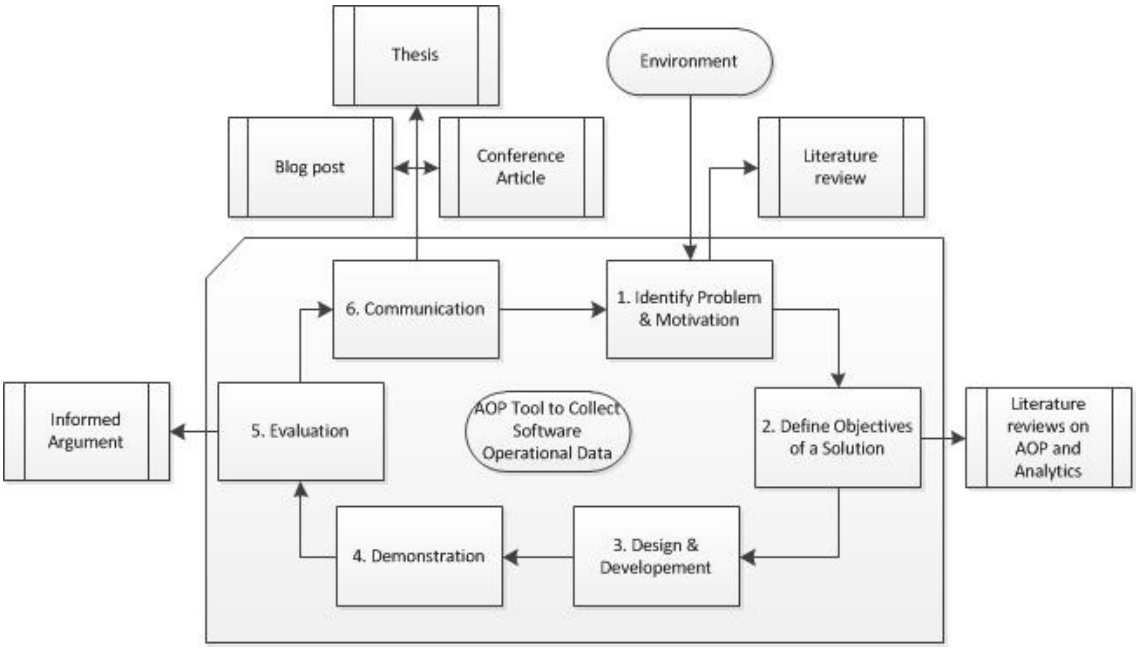

Figure 6 *Design science process with research specific methods and outcomes.*

The process starts with identifying the problem and showing its importance. The demonstrating of these was done in Chapter 1 by reviewing the literature, but the initial push for the research and its subject should be credited to the Department of Pervasive Computing and its knowledge of the technological and business environment. The methods used for the second part of the process are similar as literature reviews were conducted on AOP and software analytics as is described in Chapters 3 and 4 respectively. As Peffers et al. (2007, p.55) suggest, the objectives of this solution were inferred from the problem definition (see Chapter 1) and knowledge of what is possible and feasible (see Chapters 3, 4 and 5). These literature reviews were conducted during May, June and July of 2014.

The knowledge from these reviews is used for the third activity of the process – the design and development of the artifact. This activity took place in July and August of 2014 and the detailed description of NITCAD is presented in Chapter 6. Although all of the programming until the point of the publication of this thesis was done by the author of this thesis, a version control system called GIT (http://git-scm.com/) and a cloud based repository provider called BitBucket (https://bitbucket.org/) were used in the development. These choices were made to ease the further development of NITCAD, which is the intended to be made as open-source by Vaadin Ltd.

The demonstration phase of the process is executed by implementing NITCAD into a demo application of the Vaadin framework. This target application is described in Section 5.3. In this case, the activities of demonstration and evaluation are largely interlinked as the evaluation phase is conducted within this one demonstration and one target application and simply by the sole researcher. The evaluation method of informed

argument is intended to answer mainly the third research question set in Section 1.2. These activities were executed in October and November of 2014.

The final activity of the design science process is three-fold in this case. Firstly, this thesis covers the whole process of the research and obviously, it was written as an integrated part of conducting the research. Therefore, it was spread time-wise from May to December of 2014. Secondly, the intention of the author is to communicate some of the results also as a conference article in the near future. Finally, Vaadin Ltd. is willing to promote the ideas and results of this research in a blog post on their website.

3. ASPECT-ORIENTED PROGRAMMING

In brief, Aspect-Oriented Programming is a programming paradigm, which is developed in order to specify the various concerns of a system separately. The goal of this chapter is to develop rather a broad and general than a narrow and deep understanding of what AOP consists of and how and why it could be used in software development. To accomplish this, an introductory to AOP is given in the first section. This is then followed by slightly more profound fundamentals of AOP, and finally with some programming examples in the last subsection.

3.1. AOP in general

Filman & Friedman (2000) claim how the history of programming languages has been moving away from local and unitary languages. The first step was the introduction of subprograms, and inheritance in Object-Oriented Programming (OOP) the second. These steps were clearly improvements to the traditional procedural programming as they enabled abstraction and the separation of concerns, therefore making programs easier to develop and maintain.

However, Kiczales et al. (1997) point out that neither procedural nor OOP can answer to all the ever more complex problems of implementing important design decisions. These bits of code are forced to be scattered throughout the code leaving it tangled and again difficult to maintain. Lieberherr et al. (2001) explain this problem as follows. Implementing an operation in an object-oriented program involves usually collaboration from different classes. On one hand, if the operation is divided into separate methods on each of the classes involved, the implementation is scattered making it difficult to adapt when the operation changes. On the other hand, if the operation is put into just one method on one of the classes, too much information about the structure of the classes needs to be tangled into each such method. This makes it difficult to adapt to changes in the class structure.

According to Elrad & Filman (2001) the idea in AOP is that the various concerns of a system should be separately specified. Concerns are defined as properties or areas of interest and are said to range from high-level notions such as security and quality of service to low-level notions like caching and buffering. Some of the concerns are always neatly localized within a specific structural piece, while others cross multiple elements. AOP is trying to simplify the realization of these crosscutting components.

Elrad & Filman (2001) continue by defining aspects as mechanisms beyond subroutines and inheritance for localizing the expression of a crosscutting concern. By aggregating these crosscutting concerns into aspects and congealing them into a single textual structure, AOP makes both the aspect code and the target easier to understand. In addition, Elrad & Filman (2001) mention simpler system evolution, more comprehensible systems, adaptability, customizability and easier reuse as promises for the separation of the expressions of multiple concerns in programming systems.

To further define the goal of AOP, Kiczales et al. (1997) make an explicit distinction between two of the key terms in AOP. The properties being programmed are either components or aspects based on the following criteria. Components are well-localized, easily accessed and composed as necessary. They are specifically encapsulated in a generalized procedure. In turn, aspects are not often units of the system's functional decomposition. Rather they tend to be properties that affect the performance or semantics of the components in systemic ways. Memory access patterns and synchronization of concurrent objects are displayed as examples of aspects.

These definitions in mind, Kiczales et al. (1997) formulate the goal of AOP to be "To support the programmer in cleanly separating components and aspects from each other, by providing mechanisms that make it possible to abstract and compose them to produce the overall system." In contrast, programmers can separate only components from each other in OOP. Thus, AOP can be thought as another orthogonal dimension or an add-on to OOP. This thought is also supported by Popovici et al. (2002) who define aspects as properties that express functionality across the system. Aspects allow programmers to design a system out of orthogonal concerns providing a single focus point for modifications. In addition, Popovici et al. (2002) describe AOP as a promising technique to overcome the important concerns of modern applications like transactions, security, distribution, or logging, which are not easily expressed in a modular way. Elrad & Filman (2001) point out that it is the focus on crosscutting concerns like the aforementioned, which distinguishes AOP from previous separation of concerns technologies.

In order to implement these kinds of AOP designs, AOP programs are combined with the rest of the program. This type of combining of a program and aspects is called weaving. In addition to weaving, several other important definitions are related to AOP. Honoring the distinction between aspects and components, Kiczales et al. (1997) define component language as a tool to program the components, aspect language as a similar for aspects and aspect weaver as a tool for combining the two. The list continues with component program for implementing the components using the component language and with aspect program for implementing the aspects using the aspect languages. An application with an AOP-based implementation consists of all the aforementioned tools and languages.

In addition to these definitions, the implementation of aspects relies heavily on the following more technical attributes defined by Lieberherr et al. (2001). A principled point in the execution of a program is called a join point whereas a point cut is a means of referring to collections of join points and certain values at those join points. Advice can be attached to these point cuts and are defined as a method-like constructs.

3.2. Aspect-orientation in software development

Next, the two main concepts of AOP are presented. Obliviousness and quantification characterize aspect-orientation and therefore AOP's advantages in software development are tightly connected to these principles. Finally, the last subsection presents AspectJ as a programming language, with which one can implement AOP.

3.2.1. Obliviousness and quantification

The implementation of aspects in OOP without AOP requires cooperative programmers. With the help of inheritance this is possible from time to time, but as explained above it leads to tangled and scattered code. In a case like this, the cooperative programmers of the derivative classes are relied to invoke the desired methods of the super classes at the right time of execution. In addition to the difficulties in e.g. the maintenance of the scattered code, this approach ignores the facts that parts of the program may have already been written or the program might be otherwise out of control. Programmer's failure to be systematically cooperative is also a critical concern. (Filman & Friedman 2000)

Thus, Filman & Friedman (2000) claim requiring cooperation is not good enough. Programmers should not have to know anything additional or take any extra effort in order to make AOP systems work. This way, programmers of the base code are said to be oblivious, which is one of the two basic properties necessary for AOP. Obliviousness is meant as the original program's unawareness of the aspects. This being said, better AOP systems are more oblivious. They minimize the degree of how much programmers have to know about the aspects running over the program they have written. Programmers should be able to write their code like they would write it anyway, and aspects should be able to be added afterwards. (Filman & Friedman 2000)

According to Filman & Friedman (2000) the second of the two basic properties for AOP is quantification. In an AOP system, quantified statements are made to declare which code is executed within certain circumstances. In a broad perspective, quantification is possible over the static structure of the system and over its dynamic behavior. As the goal of AOP is defined as a pursuit to make programming expressions of the form "In programs P, whenever conditions C arises, perform action A." quantification describes what kind of conditions C are to be specified.

Filman & Friedman (2000) distinguish these two types of quantification as follows. Firstly, dynamic quantification is described as tying the aspects to the run-time behavior of the original program. Examples of such are

- raising of an exception,
- call of a subprogram,
- size of the call stack, and
- patterns on the history of the program.

The implementation and the available elements, over which to quantify vary widely depending on the underlying language. Although dynamic quantification might seem tempting, Filman & Friedman (2000) point out that e.g. in the case of exception-handling by catching remotely thrown exceptions it might be too late to do the most interesting things with them, as exception context has been lost already.

Secondly, static quantification presents two different approaches in the form of black-box and clear-box techniques. The critical difference between these two techniques is in the need of access to the base program's source code. As the AOP systems with the clear-box technique need such access, they are also more difficult to implement. On the other hand, black-box techniques allow quantification over the public interface of the base program and therefore, they are "more likely to produce reusable and maintainable aspects". Clear-box techniques quantify over internal structure, which allows access to the programs details. They can easily implement aspects associated with the caller side environment of the subprogram. (Filman & Friedman 2000)

3.2.2. Programming aspects with AspectJ

According to Kiczales et al. (2001a) AspectJ is designed to be a compatible extension to Java. This compatibility is established with four objectives. Legal Java programs must be legal AspectJ programs and these must be able to be run on standard Java virtual machines. AspectJ tools support needs to be extended for existing integrated development environments (IDE), documentation tools, and design tools. In addition, "programming with AspectJ must feel like a natural extension of programming with Java". (Kiczales et al. 2001a)

AspectJ is recognized as the de-facto language for implementing aspects and in this thesis the focus is completely on AspectJ. The main concepts seem to be very similar in each of these languages, and therefore almost everything of the following could be implemented with other aspect-oriented languages as well. Anyhow, AspectJ is selected here, as it is the most popular of the aspect-oriented languages and Java is the language of the case company's framework.

On the programming level, aspects are implemented through a joinpoint model. This model provides a reference for a program's aspect and non-aspect code to be properly

coordinated. Joinpoints are defined as well-defined points in the execution of a program. Examples of these are method and constructor calls and call receptions, method executions, field sets and gets, exceptions handler executions, and class and object initializations. (Kiczales et al. 2001a)

According to Kiczales et al. (2001b) aspect-oriented languages have three critical elements. The first one is the aforementioned joinpoint model. Secondly, there needs to be a way of identifying joinpoints. Therefore, AspectJ has pointcuts. An example of a pointcut is displayed in Listing 1. (Kiczales et al. 2001b)

Listing 1. *Example of a pointcut (adapted from Kiczales et al. 2001b).*

```
pointcut move() :
     call (void Point.setY(int) ||
     call (void Point.setX(int) ||
     call (void Line.setP1(Point)) ||
     call (void Line.setP2(Point));
```

Pointcuts can be given names and in this Listing 1, a pointcut called `move` is defined. After the name definition, pointcut designators describe the particular join points i.e. the points in the execution of a program where an aspect is invoked. These pointcut designators can be combined with logical operators like "||" and "&&". In this case, name-based crosscutting is used as pointcut designators are defined with specific class names. However, AspectJ allows also property-based crosscutting, in which a pointcut is specified in terms of properties of methods rather than their exact name. An example of property-based crosscutting is displayed in Listing 2. (Kiczales et al. 2001b)

Listing 2. *Example of property-based crosscutting (adapted from Kiczales et al. 2001b).*

```
call (void Figure.make*(..))
call (public * Display.*(..))
```

The simplest way to produce property-based crosscutting is to use wildcards in pointcut designators. In the example presented in Listing 2 the pointcut designator defined on the first line cuts the execution of a program whenever a method of Figure class with a name beginning with "make" is called. The second row states that execution is interrupted with any call to a public method defined on Display. The double dots in both of these cases make the number and type of parameters on these calls irrelevant.

As the third critical element of aspect-oriented language, a means of executing the desired code at join points is required. AspectJ answers this need with elements called advice. There are before, after, and around advices to specify when the advice code is executed relating to the join point. An example of a before advice is presented in Listing 3 on the line 2. (Kiczales et al. 2001b)

Listing 3. *Example of a before advice (adapted from Kiczales et al. 2001b).*

```
aspect SimpleTracing {
  pointcut traced():
       call(void Display.update()) ||
       call(void Display.repaint(..));

  before(): traced() {
       println("Entering:" +
            thisJoinPoint);
       }
  void println(String str) {
       <write to appropriate stream>
  }
}
```

Listing 3 provides also a complete example of an implementation of an aspect in AspectJ. On the lines 2-4 there is a pointcut similar to the one seen in Listing 1. Additionally, there is the advice code on the lines 6-9. All in all, these are combined under the aspect called "SimpleTracing" as seen on the line 1. These, in its simplicity, form all the required code parts for programming aspects.

4. SOFTWARE OPERATION DATA ANALYTICS

First in this chapter, a look is taken to define different levels of knowledge. This is crucial for the following section introducing analytics, as the concept is such a knowledge-centric one. Subsequently, to deepen the knowledge on analytics the most common technologies around the concept are studied. From there, the research focuses even more specifically on software analytics, and a look at the information needs in software development is taken. This is followed by the subject of software operation data, and an example of a mining method of that data is given. Finally, different data collecting methods for usage data are presented.

4.1. From data to insights

Themes like analytics and business intelligence are on the rise (Chen et al. 2012, p.1165; Watson & Wixom 2007, p.96). According to Watson & Wixom (2007, p.96) business intelligence is a process of two activities – getting data in and getting data out. Similarly, Zhang et al. (2011, p.55) describe analytics as the process of data exploration and analysis for obtaining insightful and actionable information.

Furthermore, it can be concluded that analytics – and knowledge in general – are valued more and more and they seem to be competitive factors for different organizations (Chen et al. 2012, p.1165). These organizations use information in making sense of change in their environment, in creating knowledge for innovation, and in making decisions about courses of action (Choo 1996, p.329). As implied, analytics can be used as a part of this process, but to better understand the whole concept of analytics, one should first understand the different levels of knowledge.

4.1.1. Data, information, and knowledge

According to Loshin (2012, p.8) the terms data, information, and knowledge are often in risk of being used interchangeably. They all have different meanings, and as a matter of fact they are not even the only terms regarding knowledge. For example, Liebowitz (2006, p.7) presents an intelligence hierarchy including five different levels, and Thierauf (2001, p.8) tops even that with his six-stepped approach. These different versions of the hierarchies regarding knowledge are illustrated in Figure 7.

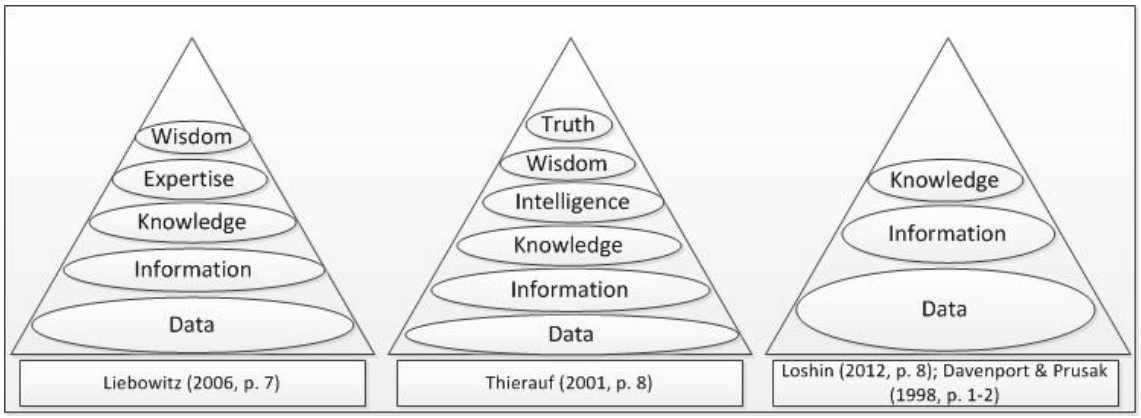

Figure 7 *Different approaches to the hierarchies of knowledge*

Although there is no single approach better than the others – even regarding the point of view of this specific research – it seems crucial to understand the essence of data turning into insights in different kinds of steps and taking different kinds of forms in the process. So despite the approach, there is data first. Data is regarded as unstructured facts (Thierauf 2001, p.8), or a collection of raw value elements without a context to put it to perspective (Loshin 2012, p.8). However, it can also be seen as structured records of transactions. Even with this description, it is worth pointing out that data by itself usually has only little relevance. (Davenport & Prusak 1998, p. 2)

Loshin (2012, p.8) describes data, and its relation to information, similarly with an example. Individual names and birthdates (i.e. data) are hardly worth anything, but when the configuration is ready on how these bits of data are to be used together, one can see a description of a party and its members been created. In the process, one has created the context for these data and turned data into a piece of information. (Loshin 2012, p.8) Liebowitz (2006, p.7) supports this view of information as data being patterned in some way. Similarly, Thierauf (2001, p.8) sees information as structured data and adds that in this form, it will be useful for analysis.

The next step in every version of the hierarchy of knowledge seen in Figure 7 is information turning into knowledge. However, the definitions of knowledge vary in some ways. Liebowitz (2006, p.7) describes knowledge neatly as information added with insights and experience. On the other hand, Davenport & Prusak (1998, p.5) specifically stress how knowledge is not neat or simple. They claim knowledge is a fluid mix of experiences, values, contextual information, and expert insight. Similarly to both of the aforementioned though, Thierauf (2001, p.8) picks out the element of experience in the description of knowledge. According to him knowledge is something that is obtained from experts and is based on actual experience. However, Loshin (2012, p.8) highlights the concept of understanding information as knowledge (Loshin 2012, p.8) and although this might involve using experience, it does not require it.

As far as this research is concerned, the term "knowledge" is perceived on one hand as a general term including all of the above terms seen in Figure 7. On the other hand, knowledge is defined as one specific instance of these terms as described above. In addition, there are a few terms in Figure 7 that describe knowledge even more profound than data, information, or knowledge. As mentioned above, the exact definitions of all of these terms are not as valuable for this research as the understanding that different states of knowledge exist and that the process order in simplicity is as Davenport & Prusak (1998, p.6) put it: "Knowledge derives from information as information derives from data".

4.1.2. Introduction to analytics

The amount of data has been increasing the last decades and it is becoming more and more crucial to know how to exploit it. At times, data is stored without further processing to databases and logs making it almost a sort of a waste. Although the necessary space for storing these data might not cost fortunes, the potential wasted in not exploiting the insights hidden within might separate a thriving business from a failing one. On the contrary, there is the data-centric decision-making of extracting and analyzing data to support informed decisions known as analytics (Baysal 2013, p.1407).

The modern tough global economy does not tolerate chance and intuition, but in these unsecure times analytics provides organizations with confidence on how to make data-driven decisions (Davenport et al. 2010). Metaphorically, the outdated way of decision making could be seen as the job of a sports referee. They watch games from their subjective point of view, they do not always have a clear view of the play, and most importantly they make decisions in an instance under high pressure and with scarce knowledge of what actually happened in the field. It is no wonder that sometimes the decisions made under these circumstances are plain wrong, and therefore there have been efforts to increase the amount of technology in order to supply objective evidence supporting the decision making. Similar to how the increasing technology is helping these sports referees to make better decisions, analytics is doing the same in the fields of business, and the amount of these fields seem numerous.

According to Chen et al. (2012, p.1166) analytics "…is often referred to as the techniques, technologies, systems, practices, methodologies, and applications that analyze critical business data to help an enterprise better understand its business and market and make timely business decisions." They continue with stating how these can be applied to several different industries such as e-commerce, market intelligence, e-government, healthcare, and security. (Chen et al. 2012, p.1166) Similarly, Begel & Zimmermann (2014, p.1) claim, that all types of businesses commonly use analytics for example to understand their customers better.

Considering the aforementioned definitions, it seems justified to conclude that analytics is seen as an umbrella term including different subjects, which as an entity aim at producing refined insights from raw data to support decision making. This kind of a view is supported by Buse & Zimmermann (2011, p.3), who state the goal of analytics is "…to help managers move beyond information and toward insight". This type of a model of analytics is illustrated in Figure 8.

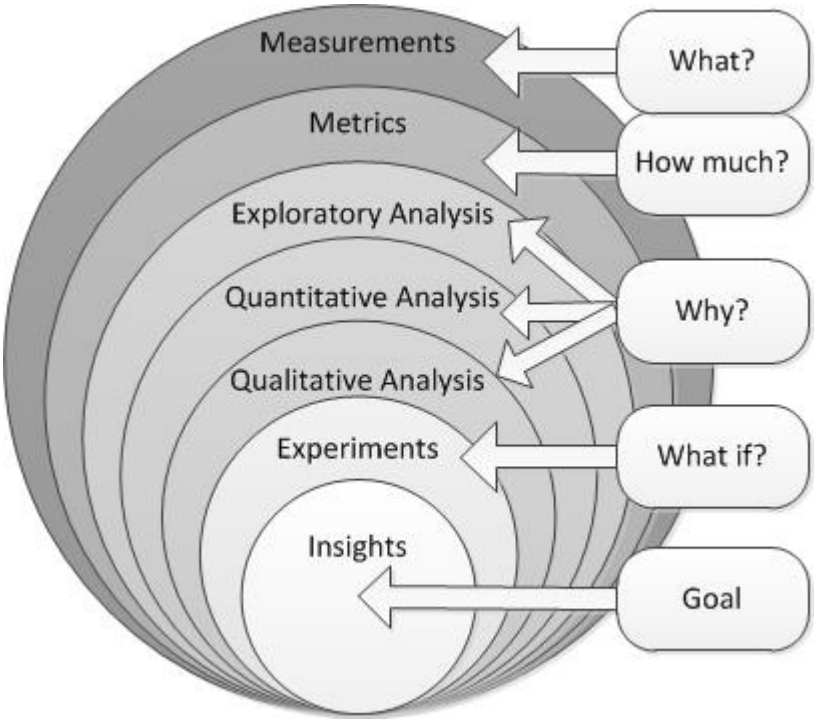

Figure 8 *Paradigm of Analytics (adapted from Buse & Zimmermann 2011, p.3)*

As is illustrated in Figure 8, analytics can be seen as a layered process which starts with measurements and continues with different kinds of analyses towards the goal of gaining insights. The increasing layers are turning simple questions of the information level such as "What happened?" to questions of insight, for example "How did it happen and why?" As this kind of a transition is not an easy one, analytics typically makes use of automated tools in these different kinds of analyses. These include for example summarizing, filtering, modelling, and experimenting – the point being that one can gather more complete insights with the help of the aforementioned instead of considering data or metrics directly. (Buse & Zimmerman 2011, p.1-3)

The benefits of using analytics seem to be as extensive as the number of industries it is used for. There are several different fields where analytics is applied to, and a few of them are presented and discussed in brief as follows.

- **Business analytics.** Sharma et al. (2010) claim that the recent interest in business analytics is due to the increasing volume of inter- and intra-organizational information systems. For example supply chain management

systems and customer relationship management systems provide users with large amounts of data, that can be further refined with business analytics tools to obtain performance gains. (Sharma et al. 2010, p.188)

- **Software analytics.** Mining user data has become increasingly important for cloud vendors in order to understand what features best attract their clients. Gaming companies apply data mining already routinely to plan the next release of their software. (Menzies et al. 2014) According to Pachidi et al. (2014) an automated usage analysis based on real execution data can lead to some beneficial business advantages e.g. improved customer satisfaction, customer retention and consequently increase in sales (Pachidi et al. 2014).
- **Web analytics.** The practice of web traffic analysis is said to have potential in improving the online user experience (Wiggins 2007). These tools are used for example to collect click-stream data in order to understand online customers and their behaviors. Ultimately the aim is to foster the users' behavior to achieve the organization's goals. (Nakatani & Chuang 2011)
- **Learning analytics.** Greller & Drachsler (2012) expect that the increase in available educational data will lead to learning analytics to support learners, teachers, and their organizations to understand and predict better the personal needs and performance of learners. Obviously, the foundational goal of learning analytics is in improving learning (Clow 2012).
- **Customer analytics.** According to Germann et al. (2014, p.1) companies in the retail industry have the most to gain from deploying analytics. They also state that prior research has shown how deploying customer analytics has had a positive relationship to firm performance. (Germann et al. 2014, p.1)

All in all, it can be concluded that analytics is a major trend cross-cutting numerous industries. However, to concentrate on the subject field of this research, the remaining sections and subsections of this chapter focus on exploring how analytics exploits data in the context of software development and turns it into insights. Firstly, a look is taken to determine some of the most common technologies software analytics consists of.

4.1.3. Technologies for software analytics process

Zhang et al. (2011, p.55) name data mining, machine learning, and information visualization as analytic technologies, with which software analytics enables the data exploration and analysis processes. (Zhang et al. 2011, p.55) Baysal (2014, p.2) has broken down software analytics in somewhat similar fashion into four areas including data mining, statistical analysis, interpretation, and leveraging. Although data mining is the only exact match between these types of software analytics breakdowns, much of the categories Baysal (2014, p.2) uses include the same technologies Zhang et al. (2011, p.55) mention. Figure 9 illustrates the process of software analytics according to Baysal (2014, p.3).

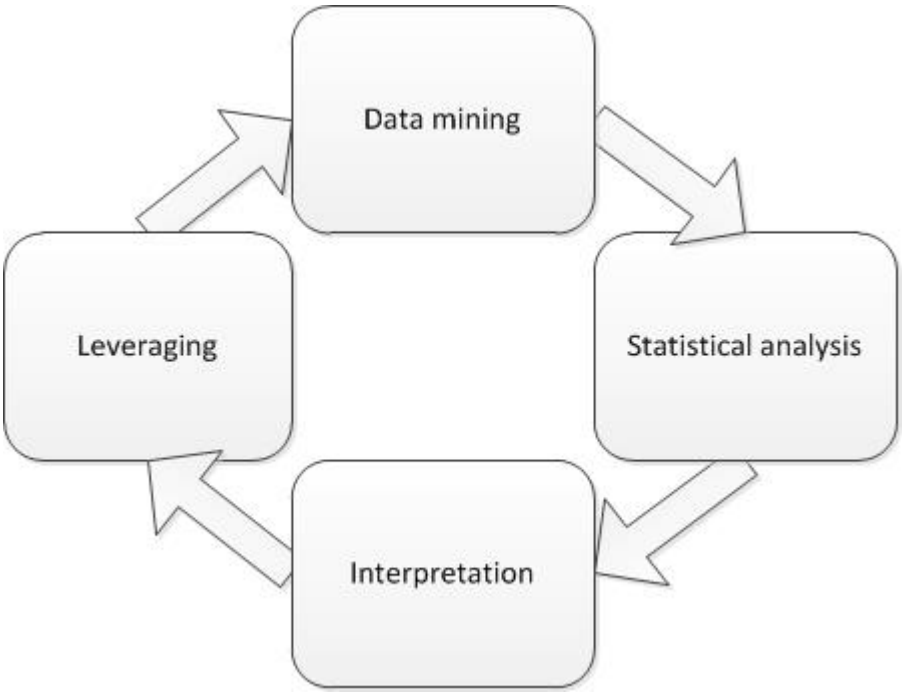

Figure 9 *Software analytics (adopted from Baysal 2014, p.3)*

Data mining can be viewed as the process of collecting and extracting information and pre-processing data for further use. This includes methods such as data cleanup, normalization, transformation, and feature extraction and selection. (Baysal 2014, p.2) As implied above, data mining in this case is both the first part of the process of software analytics as well as one technology for software analytics. Considering this latter view, Halkidi et al. (2011) have described data mining as a collection of techniques to analyze and extract novel interesting patterns from data. They continue with stating that data mining is "…the process of inducing previously unknown and potentially useful information from data collections." (Halkidi et al. 2011, p. 413)

Statistical analysis is defined in the context of software analytics as the process of exploring the data and performing analysis (Baysal 2014, p.2). In this sense, it could be interpreted as a part of data mining, given that data mining is looked at with the broader view presented above. In a somewhat similar fashion, *machine learning* could be seen as a part of it as well. Machine learning allows its users to find patterns in data more easily (Simon 2013, p.89), and with such a benefit, it is obvious to be used as an analytics tool.

Interpretation is a step of the software analytics process, which includes technologies such as *information visualizations*. Considering the hierarchies of knowledge described above, the former steps seem to be concentrating more on turning data into information and the interpretation part on turning information into knowledge. This is done with the help of presentation and visualization of findings via charts and graphs (Baysal 2014, p.2).

To conclude, the *leveraging* step is about building applications and predictive models on the basis of the former results (Baysal 2014, p.2). All in all, the whole process might turn out to be needless if the leveraging part is neglected. After all, the goal of analytics is to help practitioners make informed decisions about their projects (Baysal 2014, p.2)

4.2. Information needs in software development

Hassan & Xie (2010, p.161) describe the decision making in software development a matter of art rather than a well-studied science. This decision making consists of determining things such as when a software system is ready for release, whether a part of a software system should be re-factored or re-written, or which parts of a software system should be thoroughly tested. (Hassan & Xie 2010, p.161) However, software development can be regarded as a data rich activity with many sophisticated metrics (Buse & Zimmermann 2011, p.1). For example, Baysal (2013, p.1407) lists common information sources in software development including source code, defect data, commit history, test suits, and documentation. Although software engineers and project managers might be in lack of the necessary tools and techniques to leverage the data available, the potential is clearly there (Buse & Zimmermann 2011, p.1).

As a solution to this problem of the state of practice, Hassan & Xie (2010, p.161) propose increased use of a concept they call Software Intelligence (SI). They define SI to offer practitioners an access to fact-supported views of their software systems and enable them to produce both short- and long-term strategic planning. (Hassan & Xie 2010, p.161) In this sense, SI could be looked at as the product of software analytics process. This in mind, one of the first parts of this process should be to define the target tasks analytics should help the project with (Zhang et al. 2011, p.55). For this again, it seems inherently crucial to find out what kind of information needs there are in the first place.

Ko et al. (2007) arranged a study in which they investigated software developers for their information needs. During the observational study, 334 instances of information seeking were recorded, and these were abstracted finally to 21 general information needs. These needs are listed in Figure 10 along with the corresponding frequency and outcome of the search effort.

Of these abstracted information needs, listed in Figure 10, the most frequently sought and acquired information regarded the number of mistakes in code and what developers' coworkers had been doing.

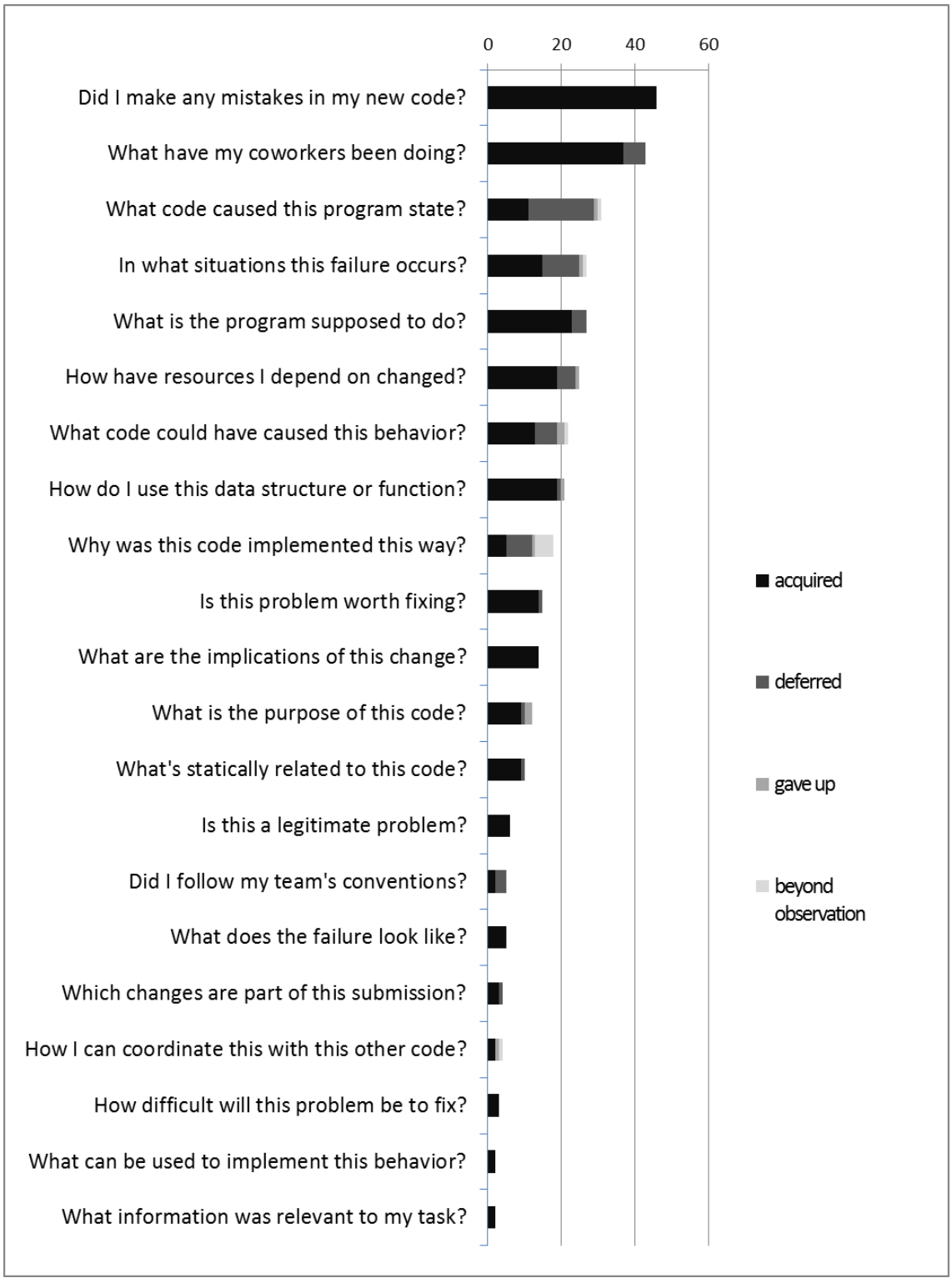

Figure 10 *Information needs of developers (adapted from Ko et al. 2007, p.7)*

More interestingly, however, the information needs, which were deferred most often, regarded the reasons for a particular program state and overall the situations in which a failure occurs. In addition, the researchers point out, that developers deferred these information searches since they depended on an unavailable person. These information

needs were also found out to be unsatisfied the most often, and this clearly hindered the productivity of developers. (Ko et al. 2007, p.7)

Moving on from the developers' point of view, according to Buse & Zimmermann (2011, p.1), there have not been as extensive studies on project managers' needs, regardless of the significant research efforts for the information needs of developers (Buse & Zimmermann 2011, p.1). This seems quite surprising as often the project managers are the ones making the important decisions about the future of projects.

To fix this lack of research, Buse & Zimmermann (2011) conducted a survey, in which also managers where invited to participate. Participants were, among other things, asked to describe decision making scenarios in software development. Of these, the researchers highlighted questions analytics could answer and decisions it could support, and came to the conclusion that approximately 89% of the scenarios concerned the past or the present, but surprisingly not the future. Additionally, they found out that while both the developers and managers described scenarios concerning testing, the rest of the managers' scenarios were wider in variety. These included refactoring, release planning, understanding customers, judging stability, training, and inspection, of which for example the second most frequently mentioned topic, release planning, was not even mentioned by developers. (Buse & Zimmerman 2011)

In addition to these, several new methods have sprung up for software development in recent years creating new kinds of information needs at the same time. These include topics such as agile methods, A/B testing, and lean startup methods.

- Agile software development is about feedback and change (Williams & Cockburn 2003). The concept is built among other things upon iterative methods which have been seen to reduce the risk of failure when compared to more traditional methods (Larman 2003). To adapt to the changing requirements, these iterative methods need to be supported with efficient communication (Fowler & Highsmith 2001).
- Lean startup is basically built around the concept of a feedback loop, called "Build – Measure – Learn" in this case. This culminates in the concept of Minimum Viable Product (MVP). As little effort as possible is tried to be used for the production of this first product, and its only purpose is to find out if the product is needed at all. (Ries 2011)
- A/B testing is a technique, which can be used in the two former, and feedback information is absolutely crucial for this one too. In A/B testing two identical software artifacts are used by real users with the exception of usually one single detail, which has been implemented in a different way (Kohavi et al. 2009). After some time of use, the developers find out which of the implementations was used in a more desired way, and they might conclude that also the rest of the implementations should be made identical to that case. Whereas the MVP is

intended more likely to test the potential of a business idea, A/B testing could be more common in finding out for example which color works best for a sign up button.

All of these concepts are based on different kinds of feedback loops, and this inherently creates the need for information specifically originating from how users use software. However, Hassan & Xie (2010, p.161) claim that the efforts have been mainly in mining code and bug repositories. Given the evolution of software development described above, the focus should obviously move also to mining the data collected run-time on how the systems are used (Hassan & Xie 2010, p.161).

4.3. Mining software operation knowledge

According to El-Ramly & Stroulia (2004, p.1) many software systems can be arranged to collect data about how users use these systems. This system-user interaction data as they call it can be valuable in understanding the systems and in reengineering purposes. (El-Ramly & Stroulia 2004, p.1) To better understand what kind of data is addressed in this case, the next subsection defines the concept of software operation knowledge. After that, a mining method for this data is presented in order to elaborate how this data could be turned into knowledge.

4.3.1. Software operation knowledge

According to Baysal et al. (2012, p.1) the techniques for mining software repositories often focus on different kinds of development artifacts such as source code, version control meta-data, defect tracking data, and electronic communication. Similarly, El-Ramly & Stroulia (2004, p.1) claim that an increasing amount of work has been done on development data (El-Ramly & Stroulia 2004, p.1). This approach lacks knowledge of in-the-field software operation however. Van der Schuur et al. (2011, p.1) point out that software vendors frequently do not recognize the potential value of this knowledge.

El-Ramly & Stroulia (2004, p.1) define system-user interaction data to consist of "…temporal sequences of events that took place while the users were interacting with the system." These include interesting patterns of both user activities as well as of the usage of system services. In this case, the researchers have used for example different screens of a legacy user interface (UI) to form navigational patterns out of this data. (El-Ramly & Stroulia 2004, p.1) An example of operation data in form of a navigational pattern, which consists of a user performing a task of retrieving information on a library item, is illustrated in Figure 11.

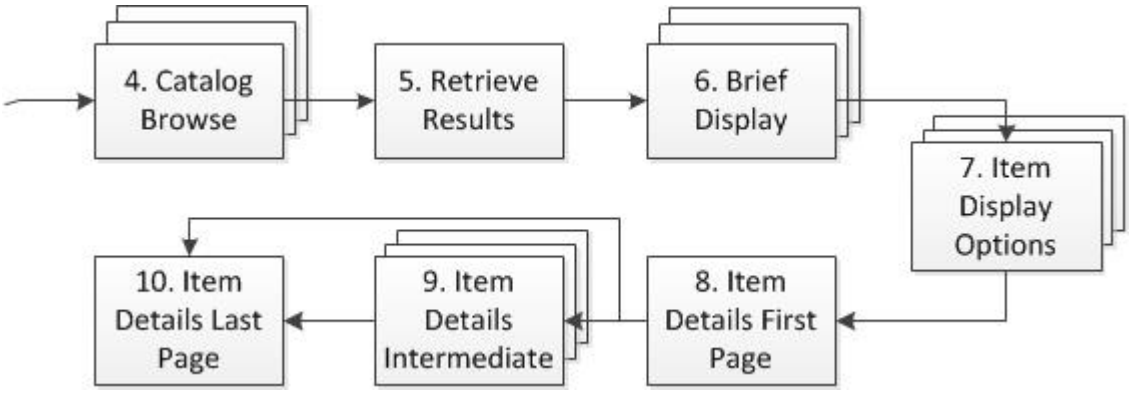

Figure 11 *Navigational pattern of performing a task of retrieving information on a library item (adopted from El-Ramly & Stroulia 2004, p.4)*

Similarly, Figure 12 illustrates the same pattern twice in form of different screens of the legacy UI. The numbers correspond with the subtasks illustrated in Figure 11. In Figure 12 the actual navigational patterns are inside the dashed lines, and the rest of the data is considered noise.

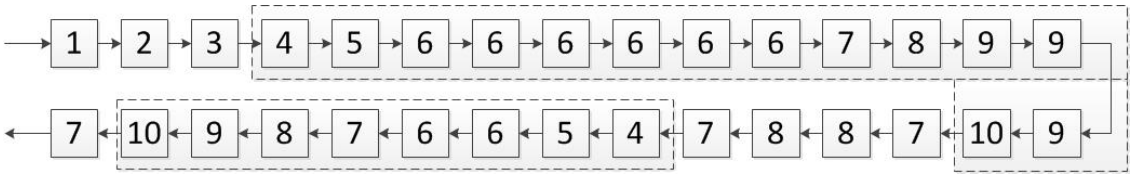

Figure 12 *Navigational data in form of numbered screens (adopted from El-Ramly & Stroulia 2004, p.3).*

The example of navigational data described above is one composition of software operation data. This approach emphasizes specifically the information of how users use a software system. However, van der Schuur et al. (2011, p.1) have also included for example data originating from the system to the term *software operation knowledge* (SOK). They define the term as follows: "Knowledge of in-the-field performance, quality and usage of software, and knowledge of in-the-field end-user software experience feedback" (van der Schuur et al. 2010, p.2). Under these circumstances, SOK is seen as much more extensive subject than mere "usage data", which nonetheless is a part of the subject.

Considering this research, SOK is set to include all these four key aspects as van der Schuur et al. (2010, p.2) have proposed. The four types are presented in the following:

A. *Performance*. Performance knowledge consists of both software performance data as well as different metrics. For example, Putrycz et al. (2005, in van der Schuur et al. 2010) have identified three types of software performance data types: device demands, interaction attributes, and logical resources. To measure performance, Johnson et al. (2007, in van der Schuur et al. 2010) mention

elapsed time, transaction throughput and transaction response time as the most common.

B. *Quality*. The ISO 9126 quality model divides software quality into three views: internal quality, external quality, and quality-in-use (ISO 9126 in van der Schuur et al. 2010). Internal quality does not depend on software operation, and therefore it will not be concerned further (van der Schuur et al. 2010). On the contrary, external quality is related to metrics such as the number of exceptions, crash report details, and mean time between failures (Bøegh 2008, in van der Schuur et al. 2010). Quality-in-use could use metrics such as end-user productivity and end-user-satisfaction (van der Schuur et al. 2010).
C. *Usage*. Software usage data consists of user interface paths, method calls, and object initiations (van der Schuur et al. 2010). With the help of different analytics tools this data could be turned into software usage knowledge, which describes the use of software in the field by its end-users and the way software responds to that use. In the case of web services, Baysal et al. (2012, p.1) pointed out how web usage data stores valuable information about users and their behavior related to adoption and use of software systems. In their study, web server logs were used as information sources formed of *hits*. These included information such as client IP address, time and date of the request, requested source, status of the request, HTTP method used, size of the object returned to the client, the referring web resource, and the user-agent of the client (e.g. browser's type and version). (Baysal et al. 2012, p.3)
D. *End-user Feedback*. Van der Schuur et al. (2010, p.3) have defined end-user feedback to consist of end-user software appreciation, criticism on certain software usage aspects, and general software experience. Metrics for end-user feedback include average feedback rating and customer satisfaction level. (van der Schuur et al. 2010, p.3)

On the other hand, Pachidi et al. (2014) have divided software operation knowledge in a somewhat different style. In their approach there are also four groups, but these consist of the following: statistical summary of sessions and users' behavior, factors that influence the customers' decisions, users profiles, and the most frequent navigation paths. (Pachidi et al. 2014, p. 585) This categorization seems to be effected by web usage mining (Liu 2006; Srivastava et al. 2000; in Pachidi et al. 2014) and anyhow this knowledge is defined to be extracted from very similar types of data as presented by van der Schuur et al. (2010, p.2) above. In this case they include usage data, user data, and corporate data (Pachidi et al. 2014, p.585-587), of which the usage data is defined much alike how van der Schuur et al. (2010, p.2) defined it. Pachidi et. al (2014, p.585-587) extend this definition with these other elements however, to enrich the out coming software operation knowledge.

The common theme in the different definitions of software operation knowledge seems to be that the original data needs to be collected in the field. Respectively, Pachidi et al.

(2014, p.583) have formed the term software usage to consist of "knowledge about how end-users use the software in the field, and how the software itself responds to their actions." The next subsection presents an example of a mining method Pachidi et al. (2014) have developed in order to understand users' behavior with software operation data mining.

4.3.2. Pachidi et al. (2014): Data mining method for usage data

Usage Mining Method, developed by Pachidi et al. (2014), turns software usage data into knowledge with several data mining techniques. The method is intended for mining software-as-a-service applications' data, which consists of the aforementioned usage, user, and corporate data. The method uses users profiling, clickstream analysis, and classification analysis as its core analysis tasks. (Pachidi et al. 2014, p.583-585) Figure 13 illustrates the process of the Usage Mining Method.

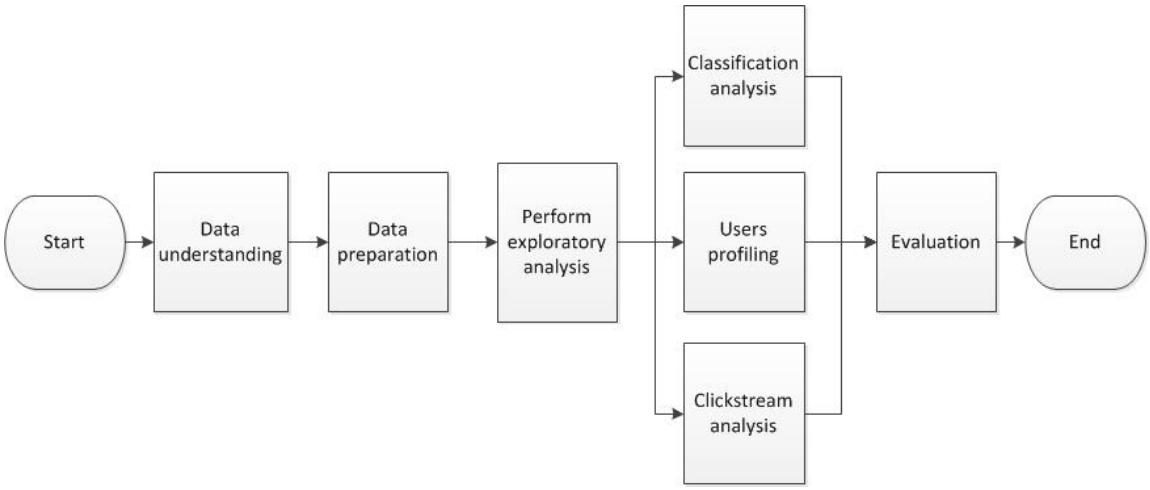

Figure 13 *Process diagram of the Usage Mining Method (adapted from Pachidi et al. 2014, p.586)*

The activities of the Usage Mining Method should be performed in the order, which is illustrated with arrows in Figure 13. This predefined order is required as the outcomes of the former activities are needed for the following step. The analysis activities, however, can be executed concurrently. The process starts with the step of *Data Understanding*. This activity includes both the selection of variables to be logged as well as logging itself. In addition, a description of the quantity and format of the data is given with an evaluation of data quality. (Pachidi et al. 2014, p. 585)

The following activity of *Data Preparation* is all about data preprocessing. This includes steps such as data selection, data transformation, data cleaning, data construction, and data integration. After these, the step of *Exploratory Analysis* produces general measures and figures with statistical analysis to be used as an input for the following data mining tasks. Until this point, the activities have had to be performed

in the described order but the following analysis tasks can be executed in an arbitrary order. (Pachidi et al. 2014, p.585)

In *Classification Analysis* a classification tree is built and evaluated with cross-validation. In *Users Profiling* several clustering models are built with different algorithms. Again these results are validated and the clustering with most suitable scores on the validation is selected. The *Clickstream Analysis* includes building Markov chains and mining sequential patterns to select interesting usage paths. Finally after these analysis steps, *Evaluation* step consists of evaluating the results of the analysis steps in terms of business success criteria. (Pachidi et al. 2014, p.585)

4.4. Collecting software operation data

This section gives an overview of the different methods for collecting software operation data. As this study is focused on understanding a novel way to collect this kind of data, a few other approaches are interesting to browse through as well. Firstly, a look is taken to find out some manual, perhaps more traditional, methods. However, as the size and complexity of software keeps on growing and at the same time somewhat more sophisticated tools become available, these processes are usually bound to be too time-consuming. Therefore, some of more automatic methods are explored briefly in the second subsection.

4.4.1. Manual methods

Even without the data revolution of the late 20th century – leading to the vast amounts of data and extensive computing power – there has always been the need to study how software artifacts are used. For example, Holzinger (2005, p.73) has listed methods for testing usability. As the most common ones his list includes *Thinking Aloud, Field Observation*, and *Questionnaires* (Holzinger 2005, p.73), which are presented as follows.

- Thinking Aloud is described by Holzinger (2005, p.73) as maybe the most valuable usability engineering method. In this method the end-user is continuously thinking out loud while using a certain software artifact. This enables the researchers to understand how users view the artifact, which again makes the identifying of the users' major misconceptions easier. For this method, time is a critical factor. Retrospectives are not as valuable as they rely on user's memory of how they experienced the use, and thus the researcher's presence in a way or another is usually required. As advantages Thinking Aloud is said to reveal why users do something, provide an approximation of how users use artifacts in practice, and give early clues for identifying sources of problems already early stages of design. In addition, lots of data can be collected from a relatively small number of users, the collected data can include vivid

explanations of the artifact, and preference and performance information can be collected simultaneously. On the other hand, as some users might concentrate better while thinking aloud, the method makes the using situation a bit more unnatural for the users. (Holzinger 2005, p.73)

- Field Observation is presented as the simplest of all methods by Holzinger (2005, p.74). In this method researchers visit users in their workplaces and take notes as unobtrusively as possible on how a software artifact is used. Any additional noise or disturbance might lead to false results. This makes video recording of the use sessions an inviting option, but on the other hand to analyze a videotape is said to be 10 times more time-consuming than of a user test. Holzinger (2005, p.74) points out that data logging is one type of means of electronic observation, and this can provide extensive timing data, which is usually important in usability studies. (Holzinger 2005, p.74)
- Querying the users can be the best method in many cases of studying usability – especially when measuring subjective things such as satisfaction of the users, and in general the things that are difficult to measure objectively. These might include things such as the preferred features of a software artifact. However, experience to design the questionnaires is required. The method collects opinions about the artifact but as Holzinger (2005, p.74) points out, these should not be trusted over the actual behavior of the user as people's claims of how they think they use software might not always be objectively the truth. As an advantage, statistical data is easy to form from questionnaires. Nonetheless, indirect methods such as questionnaires are usually low in validity and a sufficient amount of responses is needed for significance. In addition, a fewer number of problems of the software artifact can be identified than with other methods. (Holzinger 2005, p.74)

These methods are probably going to retain their value in the future in delivering understanding especially on how individual users use software artifacts. However, Hertzum & Jacobsen (2001) have found that usability evaluation methods such as cognitive walkthrough, heuristic evaluation, and thinking aloud suffer from substantial evaluator effect. This means that different evaluators evaluating the same user interface with the same method are getting different results (Hertzum & Jacobsen 2001). Additionally, the outbreak of excessive computing power and the new means of processing data call also for more automated and sophisticated methods in collecting software operation data.

4.4.2. Automatic methods

There are several ways of collecting software operation data in an automated manner. However, to each of these different means there are both advantages as well as disadvantages. These are presented as follows:

- *Application specific data gathering systems*. Basically any software artifact can be designed to keep some kind of a log of how it is being used and what is happening in its execution flow in general. In general, there are no reuse options for this approach however, and every new case needs a new implementation. In addition, there is the major risk of leaving the source code tangled. As the monitoring snippets of code are needed in many places of the original source code, maintenance and debugging might become substantially more difficult.
- *Collection framework*. This approach on the contrary has the reuse option as its strength. This leads not only to the easier implementation of the data collecting tool but also to that the data gathered from different sources are comparable among themselves. Consequently, the same analysis tools can be used on differently originating data even without further configuration. Unfortunately, upfront decisions are needed when designing the software artifacts to be suitable for the collection framework. Therefore, difficulties may arise when this approach is applied on legacy software.
- *Instrumentation with tools similar to those used in testing*. As was the case with collection frameworks, reuse options are inherently available and data is comparable between different sources. Commercial off-the-shelf (COTS) tools might lead to excessive instrumentations however, and this might turn out harmful for performance. When compared to the application specific tools, these COTS tools lack the same kind of customizability. This obviously restricts the types of data one wants to collect, and application specific tweaks are commonly needed.
- *Aspect-oriented design*. As the final approach presented here, aspect-oriented design is of course the one being studied the most in this research. Overall, it seems that there are quite many advantages in this approach including: the separation of concerns related to the operation data collecting; reuse options are available even for legacy software with just replacing the system specific hooks; data can be comparable between different implementations; evolution of the original software artifact can go on as no injections are needed.

5. RESEARCH CONTEXT

The purpose of this chapter is to give the reader an adequate knowledge of the case company Vaadin Ltd and its open source Java web-framework. The first section covers Vaadin as a company, short look at its history, its mission and vision, and how these can be seen in the design of the Vaadin Framework. In the second section the Vaadin framework is presented. This section uses the Book of Vaadin (Grönroos 2013) as the main source of information. As this book is the company's own product and intended as a reference guide, it obviously covers the framework in its finest detail, and as in this case the knowledge gained is purely technical, the problems of source criticism are considered minimal. Finally, the last section presents the target application, of whose side the developed aspect-oriented NITCAD is woven to. Similarly, this section uses mainly one source for its information – in this case the open-source source code available in https://github.com/vaadin/dashboard-demo.

5.1. Case company presentation

Vaadin Ltd was founded back in the year 2000 as IT Mill (Yritys- ja yhteisötietojärjestelmä 2014). The Finnish company's "…team had a desire to develop a new programming paradigm that would support the creation of real user interfaces for real applications using a real programming language." (Grönroos 2013, p. 9) Called the Millstone Library at the time, this first version was used in an application for pharmaceutical industry already in 2001. Ever since, the company has implemented numerous business applications with the library. Along the way, an AJAX-based presentation engine was introduced and the client-side rendering was rewritten with Google Web Toolkit (GWT). (Grönroos 2013, pp.9-10).

A major decision was also to go open-source. The library's 5th release was published under the Apache License 2 in 2007, and with the 6th release in 2009 the number of developers using the library is said to have exploded. At the same time the library was renamed from IT Mill Toolkit to Vaadin Framework, which is now in its 7th release. (Grönroos 2013, p.10) It seems that the company has focused more and more to the development of this framework as they have presented their vision as follows: "Vaadin is the most used web framework for building business applications" (Great Place to Work® Institute, Inc. 2014). Being the flagship product of the company (Grönroos 2013, p. xiv), IT Mill changed its name accordingly to Vaadin. As the primary technology of the company is free to use, Vaadin Ltd offers "Vaadin Pro Tools", training, "Vaadin Certification", support and consulting services as its commercial offering.

The company's technologies are used by over 100 000 developers in +150 countries (Great Place to Work® Institute, Inc. 2014), and as the revenue and number of employees have increased each year from 2009 (Sadeoja 2014), the decision to publish under an open-source license seems to have been justified. The detailed economic numbers are presented in Table 2.

Table 2 *Economical data of Vaadin Ltd (Sadeoja 2014)*

	2009/12	**2010/12**	**2011/12**	**2012/12**	**2013/12**
Revenue 1000 EUR	1850	2262	3060	3469	5346
Revenue change %	34,60	22,30	35,30	13,40	54,10
Profit 1000 EUR	2	15	125	3	115
Profit %	-0,20	0,60	3,70	0,10	2,90
Employees	34	-	45	56	65

Vaadin Ltd states its mission as follows: "Make building amazing web application easy" (Great Place to Work® Institute, Inc. 2014). This can be considered to be nicely in line with the framework technology being open-source and the company offering the support for its use. Additionally, the aforementioned number of developers using the framework speaks for itself but of course the number has not peaked out on its own. According to Grönroos (2013, p.8-9) the framework has been designed with the following rules, which all seem to be quite developer-based.

- Right tool for the right purpose.
- Simplicity and maintainability.
- XML is not designed for programming.
- Tools should not limit your work.

Firstly, the framework has been designed for developing web applications – not websites. Therefore, one might find smaller-scale tools more suitable for some of the less complex tasks at hand. Secondly, robustness, simplicity, and maintainability are said to be emphasized. (Grönroos 2013, p.8) Considering the way Vaadin Ltd has built the framework this guideline probably holds to its promises, but as Maple (2013) points out, specially the code generated automatically with the graphical tool might turn out to be difficult to maintain. Thirdly, the framework is developed to free programmers from the limitations of the document-centered view of the Web and the declarative presentation of user interfaces. Finally, the framework is designed in such a way that it should not limit the creativity of its user. For example, adding new, re-usable, and easy

to maintain user interface components is said to be easy with the framework. (Grönroos 2013, p.9)

5.2. Framework of the case company

Vaadin Framework is a Java web application development framework for the creation of web-based user interfaces. Being a Java framework, it enables its users to develop web applications in Java without the knowledge of browser technologies like HTML and JavaScript. There are two different approaches to develop with the framework: server-side and client-side. The server-side programming resembles largely development of a desktop application as the framework is responsible for the events of user interface in browser and the communications between the browser and the server. This architecture of the client- and server-sides is illustrated in its simplest form in Figure 14.

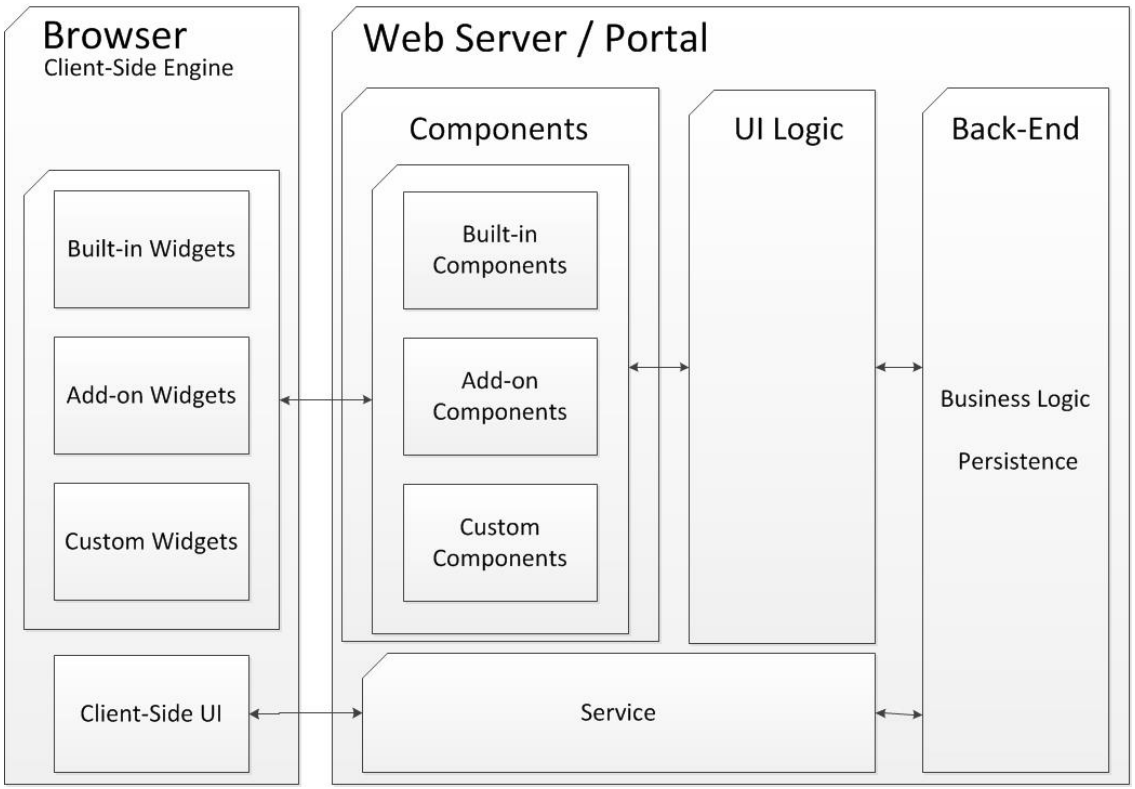

Figure 14 *Vaadin Application Architecture (Grönroos 2013, p.4)*

The server-side Vaadin applications render the server-side components with the client-side engine illustrated in Figure 14. These applications are run as Java Servlet sessions in a Java server, which handles the HTTP requests. This communication between client and server is enabled with AJAX (Asynchronous JavaScript and XML, see http://www.w3schools.com/ajax/) techniques. Although the server-side development is seen as the more powerful one, the client-side offers also development of widgets and applications in Java, which is then compiled to JavaScript and afterwards executed in

browser. One does not have to limit the development work to only either server- or client-side however, but the two types of development approaches can be mixed together. For example, the two can share their UI widgets or back-end code.

In the following Subsections 5.2.1 and 5.2.2 a closer look at the different development approaches of client- and server-side is presented. This is to deepen the understanding of the framework's architecture before jumping into the more technical description of the target application in Section 5.3.

5.2.1. Server-side development

Server-side application architecture of the Vaadin Framework is illustrated in the following Figure 15. The Client-Side Engine communicates with the server-side through HTTP and AJAX requests. On the server-side, Vaadin applications use the Java Servlet API, which in this case is the VaadinServlet class. It determines the right Vaadin Session based on cookies and delegates it correspondingly.

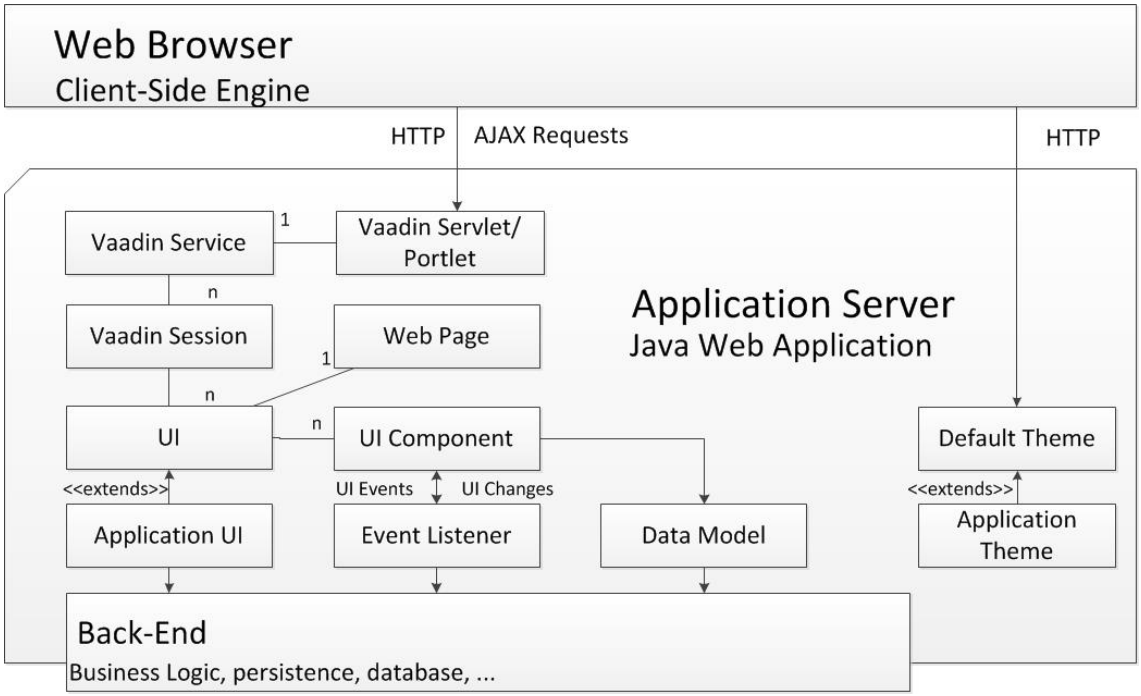

Figure 15 *Server-side application architecture (adopted from Grönroos 2013, p.66)*

The UI class is a center piece in the Vaadin Framework architecture. Every Vaadin application must have at least one of these extending the abstract com.vaadin.ui.UI class and implementing its `init()` method. Typically, a UI representing an HTML fragment fills the entire page, but as implied above there might be multiple UIs for one Page as well. The Page object represents the web page and/or the browser window, and the current Page object can be accessed with `Page.getCurrent()` method similarly to how the current UI is accessed with `UI.getCurrent()` method. Even if a user opened up several browser windows at once, the Web Pages representing them will be

associated with only one Vaadin Session. As is illustrated in Figure 15, the Web Pages are linked with the Vaadin Session only through the UIs, and not straight together with one to one relationships.

User interface of a Vaadin application is created with placing UI Components hierarchically to the UI. User interactions with these UI Components launch UI Events, which are handled by the application. These Event Listeners then update both the Back-End as well as the associated UI Components. Most of the UI Components have their own Event classes and the corresponding Event Listeners. For instance for a Button, which is a UI Component, there are `Button.ClickEvent` events, which again are listened for with the `Button.ClickListener` interface. Listing 4 presents an example of a button's and its listener's initializations.

Listing 4. *Example of a button initialization (Adapted from Grönroos 2013, p.60)*

```
final Button button = new Button("Push it!");
button.addClickListener(new Button.ClickListener(){
     public void buttonClick(ClickEvent event) {
          button.setCaption("You pushed it!");
     }
});
```

In addition to getting and setting data through Event Listeners, Data Binding can be accomplished also through field components, which are essentially UI Components for inputting values. In this case, the data is bound more directly from the UI Components to the Back-End, and the field components are more like views to data represented in the Data Model. For example, a Table component (UI Component) can be bound to a SQL query response. That way, no additional control code is needed for accessing and updating the data as a field component is bound to a property and a row to a group of fields.

The server-side architecture separates the presentation from the logic of the user interface. The presentation is defined with themes, which are typically CSS or SCSS although custom themes can be added also as HTML templates. As a starting point, Vaadin provides Default Themes, which are extended to Application Themes in running applications.

The user interface created in the server-side of a Vaadin application is rendered by the Client-Side Engine for a browser. The Client-Side Engine is basically a widget set as it mainly consists of *widgets*; a GWT name for user interface components. These are paired with their server-side counterparts, with which they connect through a specific Connector using asynchronous HTTP or HTTPS.

5.2.2. Client-side development

As implied with the description of the Client-Side Engine, the Vaadin Framework uses GWT as its basis for the client-side development. The idea behind this approach is to enable the development of client-side components in Java, and compile them into JavaScript with the GWT compiler. Vaadin extends the abilities of the GWT compiler with its Vaadin Client Compiler. These client-side components can be integrated with the server-side as was described above, but the Vaadin Framework enables them to be used also on their own as pure client-side GWT applications. After the compilation, the code is run in browser as JavaScript. The following Figure 16 illustrates the architecture of client-side Vaadin applications.

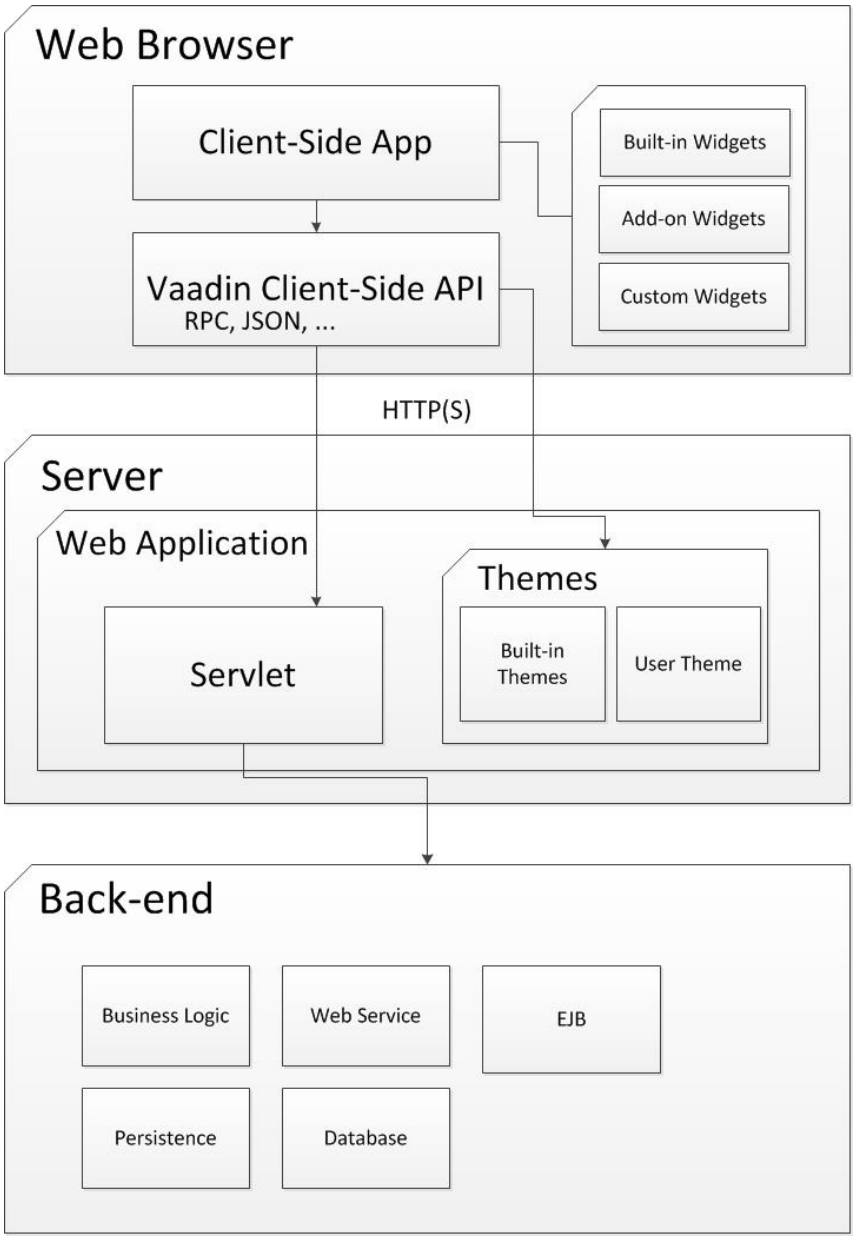

Figure 16 *Client-side application architecture (adapted from Grönroos 2013, p.447)*

Client-side applications do not need to be connected with the server-side, but they can be run on their own in a browser. However, when combined with a server-side service the applications become so called fat clients. This separates them from the thin clients constructed with the Vaadin server-side development approach. These fat clients are specifically useful when highly responsive user interface logic is needed, which could be the case for example in game applications. The same back-end services and themes used by server-side applications can also all be used by client-side applications.

5.3. Demo application of the framework

In order to see how aspects can be used for collecting data, a target application had to be chosen for NITCAD to be woven into. The criteria for the selection were that the target application should be easy to understand, ready to deploy, and it should showcase the basics of the Vaadin Framework. As an open-source product, it was no surprise there were some demo applications of the framework freely available too. Of these, an application called QuickTickets Dashboard fulfilled the set criteria most extensively. Released on January 24, 2013, QuickTickets Dashboard is the latest demo application showcasing what one can do with the framework in general. This demo application was developed by Vaadin Ltd to be the current go-to demo of their framework. (Koivuviita 2013)

Originally the QuickTickets Dashboard was developed for a Vaadin scalability study presented by Pöntelin (2011). Vaadin wanted to study how an application developed with their framework scaled performance-wise as the stress by the users goes up, and so they created an application, which works a fictional movie ticket box office. As the application is intended for selling tickets all over the world, it should be able to handle thousands of concurrent requests. (Pöntelin 2011)

Unlike the scalability study's application, which was more of a customer interface for purchasing movie tickets, the QuickTickets Dashboard was developed into an administrative dashboard type of web application. This was more in line with the purposes of the applications normally developed with Vaadin Framework, and therefore, it should better serve as a showcase application for the framework. (Koivuviita 2013) Figure 17 illustrates the user interface of the application as a screen capture.

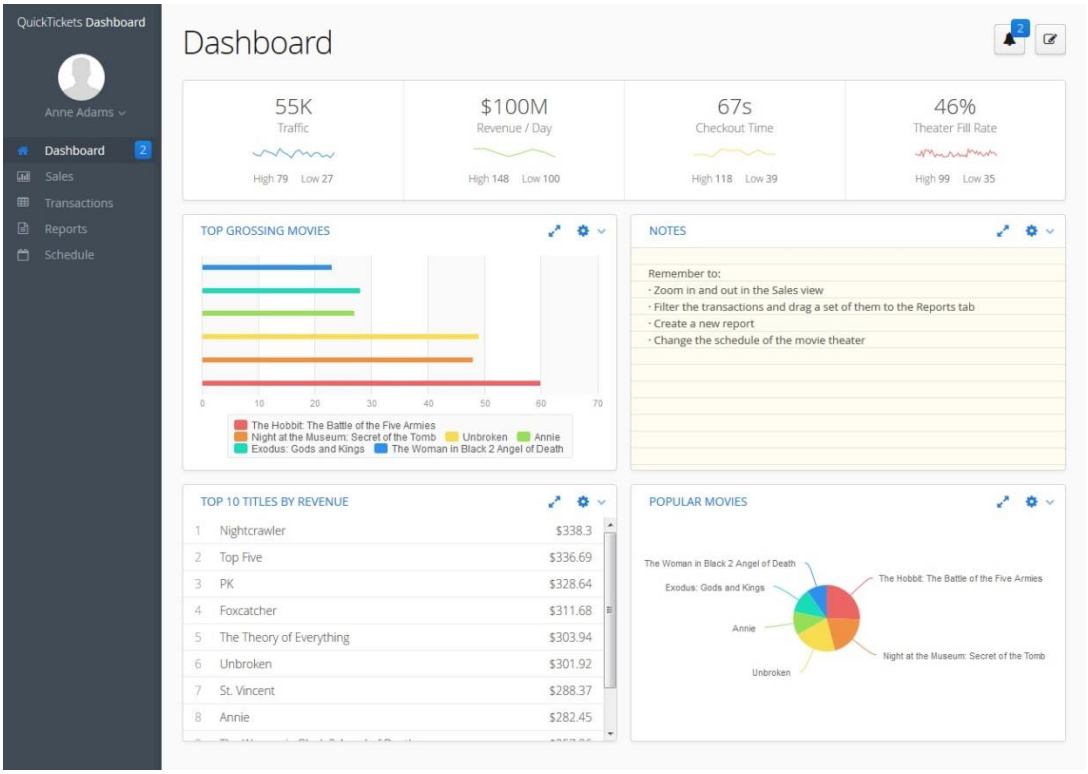

Figure 17 *Screen capture of the QuickTickets Dashboard's user interface (http://demo.vaadin.com/dashboard)*

A fully functional application is available at http://demo.vaadin.com/dashboard. When considering the use of the application, it first asks the user to sign in with a username and a password. There are actually no usernames or passwords, and simply a click of the sign in button is enough in this case. This takes the user to the main dashboard view of the application illustrated in Figure 17. On the left there is the general navigation toolbar, which includes links to the different pages of the demo application. These are dashboard, sales, transactions, reports, and schedule. The navigation toolbar is the same in each of these pages. Additionally there are buttons for settings and exiting the application in the lower-left corner. The dashboard page consists of four summarizing screens: top grossing movies, notes, top 10 titles by revenue, and popular movies. On the upper-right corner there are buttons for notifications and for editing the "My Dashboard" header. Although each of the pages listed in the navigation bar is fully functional, their purpose is only to demonstrate what the Vaadin Framework is capable of doing. For this research, their existence as working links to different pages of the web application is adequate enough, and therefore their functionalities will not be explained further.

Considering the scope of this research, a more interesting aspect of the QuickTickets Dashboard is to inspect how it has been built. Architecturally, the application is much like any Vaadin application developed with the server-side approach. It has a Java class called `DashboardUI` extending the basic `UI` class. This implements an `init` method, through which the rest of the execution follows. Additionally, the different

pages of the application listed above are technically `View` classes – UI components extending other UI component classes like `VerticalLayout` and implementing the basic `View` class. This means that within this demo application, the Web Page stays all the time the same and only the URI fragments change according to the requested `view`.

The structure of this Web Page is built with adding `HorizontalLayout` and `VerticalLayout` objects with `addComponent` method to the `root` element, which is basically the first `Layout` component added to the page. These different layout components can be on one hand nested (i.e. added inside each other) and on the other hand removed when necessary, forming always the structure of a requested view. An example of adding a new layout to the `root` element is presented in Listing 5.

Listing 5. *Adding a sidebar to the root element (lines 218-304 of DashboardUI.java).*

```
root.addComponent(new HorizontalLayout() {
      {
            setSizeFull();
            addStyleName("main-view");
            addComponent(new VerticalLayout() {
                  // Sidebar
                  {
                        …
                  }
            });
            …
      }
});
```

The more specific UI Components, for instance `Button` objects, are added in a similar fashion. However, they are not added straight into the root element, but to the specific layout component inside the hierarchical grid of these layout components. An example of this is presented in Listing 6.

Listing 6. *Adding an "exit" button to the navigation bar (lines 249-294 of DashboardUI.java).*

```
addComponent(new VerticalLayout() {
      {…
            Button exit = new NativeButton("Exit");
            exit.addStyleName("icon-cancel");
            exit.setDescription("Sign Out");
            addComponent(exit);
      …}
});
```

6. RESULTS

Firstly, this chapter presents the aspect-oriented tool for collecting software operation data from the demo application of the Vaadin framework. This presentation is divided into two subsections, which describe NITCAD's implementation and functioning as well as the collected data types. This experimental part of this research has an inductive approach, and therefore the following section starts with a genuine example of how an aspect was implemented as a part of NITCAD into the demo application. A view of how NITCAD works is presented next with the help of a sequence diagram and this is followed by a description of what types of data NITCAD collects. Secondly, NITCAD is evaluated in terms of informed argumentation.

6.1. NITCAD: The developed tool for aspect-oriented data collecting

The demo application of the Vaadin Framework was first downloaded from a web-based repository hosting service Github, and then imported as a Vaadin project to integrated development environment (IDE) called Eclipse. This IDE automatically compiles Vaadin projects and gets them up and running in local environment by setting up a server at localhost. After manually checking from a web browser that the demo application was actually working, the project was turned from a Vaadin project into an AspectJ project in Eclipse. This made no alterations to the source and was mainly done automatically by Eclipse but in addition, a new dependency had to be inserted manually to the pom.xml file of the demo application source code. This insertion was the only one in the whole process.

After turning the demo application into an AspectJ project, Eclipse had it ready and compiled the rest of the way. As aspects were then developed, Eclipse wove them into that original program. These developed aspects and the combination of them with the original program is described in the following.

6.1.1. Listening to the execution flow of a target application

The starting point of implementing any aspect is to decide on the points of execution, where the desired insertions are intended to be made. In terms of AOP, the joinpoints are first defined with pointcuts, and then advice code is woven to be executed in these places. In this case, adding of a button component to the user interface was selected as a common joinpoint, since buttons can be used as a natural way of recognizing interactions from users. As a button is clicked, something is almost certainly expected to

happen. Additionally, buttons usually provide a method for adding some kind of listeners for their clicking. Fortunately, the Vaadin Framework is no different in this sense and the click listening feature was available for its buttons as well. This was used extensively for NITCAD, and an example of its use is described in the following.

One of these joinpoints of adding a button can be identified by looking at the source code of the demo application on the lines 43-290. These are listed in Listing 7. In this case, the source code defines firstly a new `VerticalLayout` class called `DashboardView`. This is depicted in Figure 18 within the red rectangular. Secondly, a `HorizontalLayout` element is created inside the `DashboardView` class. This element, which is stored in the variable called "top", is seen within the green rectangular in Figure 18. Thirdly, within that element there is a button marked with a blue rectangular. This `Button` is stored in a variable called "notify" as the listing 7 shows. After its initialization, an ordinary `ClickListener` is added to wait for user interactions with it. Finally and as the most critical point for NITCAD, the notify button is added to the top element with the `addComponent` method.

Listing 7. *Button added to a toolbar, which is added to a view in DashboardView initialization. (Lines 43-290 of DashboardView.java)*

```
public class DashboardView extends VerticalLayout
implements View {
      …
      public DaschboardView() {
            …
            HorizontalLayout top = new HorizontalLayout();
            …
            addComponent(top);
            …
            Button notify = new Button("2");
            …
            notify.addClickListener(new ClickListener()  {
                  …
            });
            top.addComponent(notify);
            …
      };
      …
}
```

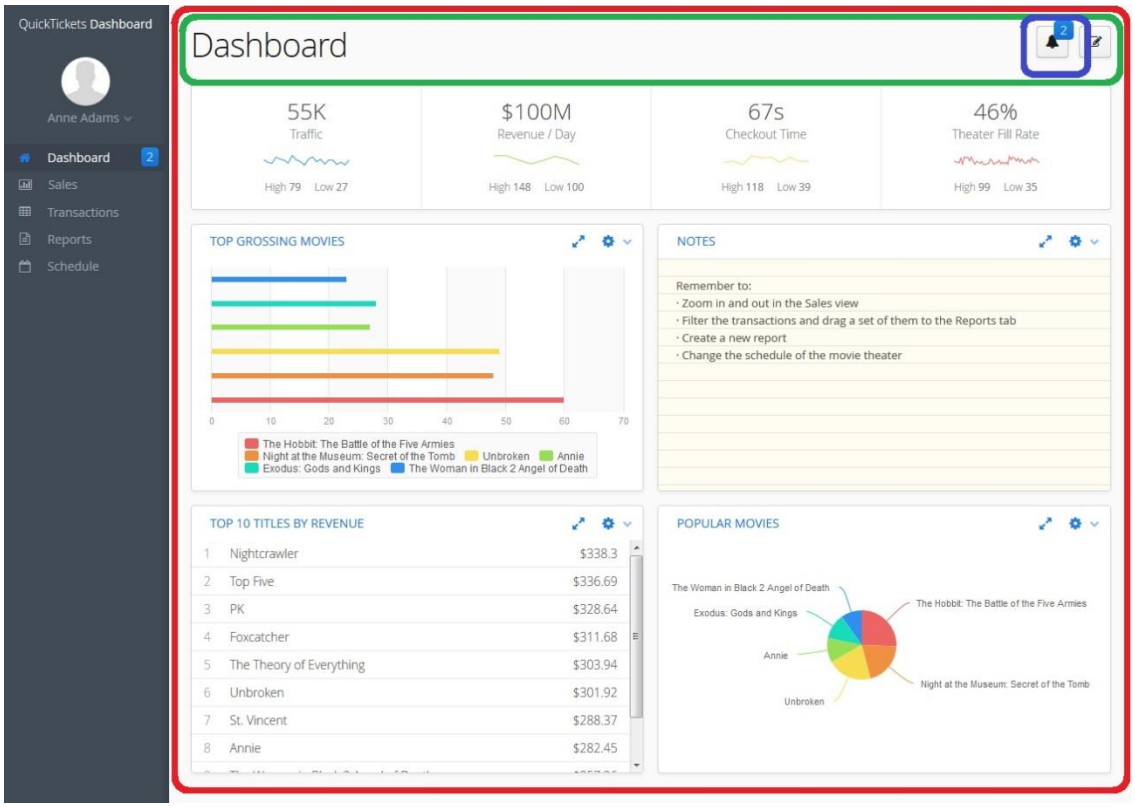

Figure 18 *Vertical and horizontal Layout components, and a button in the demo application*

The `addComponent` method is crucial, as the decision for a joinpoint was to attach additional functionalities specifically to the point where a button is added to the layout. With this joinpoint defined, a look is now taken to clarify the necessary pointcut correspondingly. Listing 8 shows how a pointcut to this kind of a joinpoint was implemented in AspectJ language.

Listing 8. *Implementation of a pointcut for adding a button component.*

```
public aspect AddComponentListener {
      …
      pointcut addComponentCall( Button b ) :
            call(* *.addComponent(*)) && args( b );
      …
}
```

This pointcut called `addComponentCall` defines that wherever a method call of `addComponent` is made with a `Button` as its argument, the execution of the program is interrupted. This should also clarify the difference between joinpoints and pointcuts. The pointcut is sort of a regular expression which is used for defining the kind of situations where the insertions are intended to be made. In this case, the pointcut is defined in such a way that a call for any type of a method (*) for any object (*) with the method `addComponent` and a `Button` object as the one argument is accepted.

After the definition of the pointcut, the functionalities to be inserted are defined as advice. These can be added either before, around, or after the execution has been cut. In this case, the functionalities of NITCAD interrupt the execution after the call defined by the pointcut is already made. Listing 9 presents the detailed source code of the advice.

Listing 9. *Implementation of advice for adding an additional click listener.*

```
public aspect AddComponentListener {
      DataLogger aoDataCollector = new DataLogger();
      …
      after( final Button b ): addComponentCall( b ) {
            System.out.println("BUTTON ADDED TO LAYOUT");

            b.addClickListener( new Button.ClickListener() {

                  public void buttonClick(ClickEvent event) {
                        aoDataCollector.logButtonClick(b);
                  }
            });
      }
}
```

In other words, the aspect defined in Listings 8 and 9 cuts the execution flow of the demo application whenever a button object is being added. After the original call is made, the aspect adds an additional click listener to the button which was being added by the demo application. After that addition, the aspect allows the original execution continue as if nothing had happened. Figure 19 illustrates this flow of events as a sequence diagram.

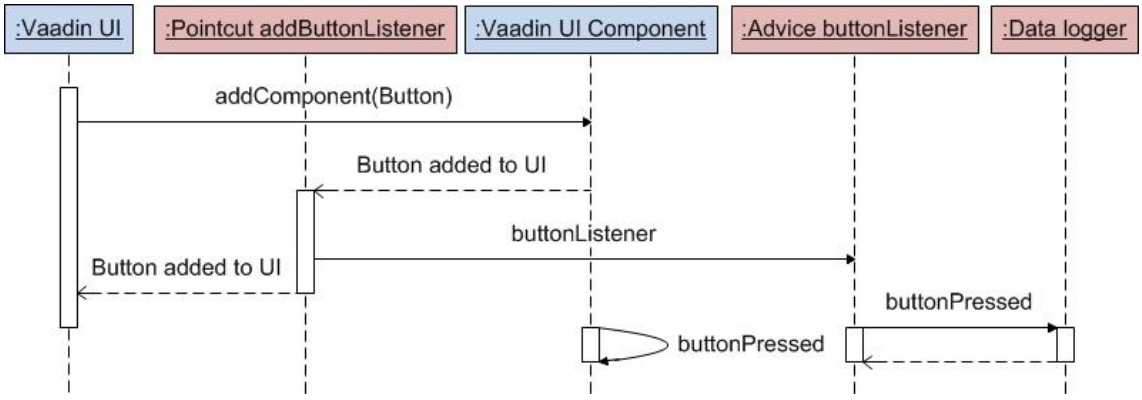

Figure 19 *Sequence diagram of a button being added to Vaadin UI, and the same button being clicked.*

In Figure 19 the components of the demo application are marked with blue and components of NITCAD with red. As can be seen, the aspect-oriented tool does not

affect the execution of the demo application. The `AddComponentListener` aspect simply listens to the execution flow of the target application and after it notices that a `addComponent(Button)` call is made, it adds its own `buttonListener` on top of the original application.

After this, the same button has in most cases two separate click listeners – one from the demo application and one from the aspect. These are also seen in blue and red correspondingly in Figure 19. As the button is clicked, both of these click listeners do whatever they have been programmed to do. In the case of the "notify" button, the one from the demo application shows a pop-up of notifications and the other from the aspect stores the information of the button being clicked as well as other related information. The following subsection looks deeper into this information.

6.1.2. Types of information collected

After the insertion of a button click listener in aspect-oriented fashion, the rest of the information collecting is done in plain Java code. Specifically, a Java class called `DataLogger` was developed for the job. The full source code of the `DataLogger` class is presented in Appendix 1, but the following list summarizes the most crucial types of information the class was set to record from each button click.

- **Session identifier.** The Vaadin Framework forms automatically a session identifier for every new user session. Based on the use of NITCAD in the local development environment, these identifiers were strings of 32 characters and numbers. A new session identifier was created if the demo application was accessed with a new browser, but if only a new tab in same browser was used for accessing, the session identifier stayed the same.
- **URI fragment.** The Vaadin Framework provided an access to a currently viewed Page object. Through this object, the current universal resource identifier (URI) fragment was able to be accessed and recorded. These fragments consisted of the last part of the URI, i.e. parts after the slash following the domain name. A thing to consider was that the currently viewed Page object was literally the current page. This became critical in situations where the button worked as a link between two pages. As the recording of a URI fragment was done after clicking a button, the currently viewed page was already changed to a new one. Therefore in most cases, the URI fragment recorded was the page where the button was set to link to.
- **Button caption.** This information was a straightforward choice to be collected as it makes the first interpretations of the collected data fairly easy for human eyes. However, the captions seemed to change rather dynamically, and thus the information provided by this `getCaption()` method might not lead to as constant results as desired by some standardized data mining methods.

- **Button identifiers.** On the contrary to the button caption information, the button identifiers seemed to provide a more constant way of recognizing buttons. First off, a method called `getId()` was tried out for identifying buttons but it seemed that this method returned an identifier, which could have been set by developer (i.e. the button's implementer). In the case of the demo application, these identifiers were not set and therefore, no data identifying buttons could be gained with the method. Fortunately, the Vaadin Framework provided also another method called `getConnectorId()`. This method returned Integers, which seemed to be unique for each button and still constant from a session to the next.
- **Request duration.** The Vaadin Framework provided an access to a `VaadinSession` object through which a method called `getLastRequestDuration()` returned the time spent servicing the last request in the current session in milliseconds.
- **Request timestamp.** Similarly to the request duration, the request timestamp information was provided through a `VaadinSession` object. `LastRequestTimestamp()` returned the time when the last request was serviced in the current session. The time was presented in milliseconds since the epoch.

The types of information described above form a summary of the information seen as the most crucial ones. During the study however, several other types of information were recorded in addition to them and these are presented in Appendix 2.

Part of the `DataLogger` class is an `IndexedContainer` variable, in which each button click along with its details is stored. This `IndexedContainer` class is part of the Vaadin Framework and it works as sort of a structured query language (SQL) table. The framework binds its data conveniently to any given `Table` object, and therefore the presentation of the collected information is relatively easy. Future development of NITCAD in mind, the decision for a SQL type of a table should simplify the further processing of these pre-organized pieces of information. In comparison, if the information about the clicks and their details was plainly logged to a text file, making it basically just bits of data, the upcoming data mining could get somewhat more complex.

6.2. Informed argument

This section evaluates the just presented tool in terms of how it achieves to deliver the benefits of AOP paradigm and how it is able to produce the information needed for software analytics. To accomplish this, the benefits of AOP paradigm and the information needs of software analytics from the chapters above are presented in a summarized form first in the respective subsections. The tool is then evaluated by the researcher based on these highlighting characteristics and expectations. Along with this

evaluation some generalizations are formed based on the specific case setting and aforementioned literature reviews. Similarly, some speculations by the researcher are presented on how the approach could work also outside this case.

6.2.1. NITCAD evaluation: Benefits of the AOP paradigm

The crux of the AOP paradigm is in the separation of concerns, which should result in various benefits such as easier maintenance and understanding of both the aspect and the original source code. AOP allows programmers to design their programs out of orthogonal concerns providing a single point for modifications. This results among other things in the non-existence of the need for modifying the original source code.

Fortunately, the aspect-oriented NITCAD developed for this research was similar in this sense. The implementation in AspectJ used a clear-box technique, which on one hand required an access to the source but on the other hand did not demand any alterations to it. This non-invasiveness might prove to be quite vital in some cases when the original application is already developed or if it is out of reach in some other way. This of course was the case also here as the demo application was already written, and still it was possible to add the collecting functionalities on the side.

The benefits of this non-intrusive way of adding functionalities to programs will not stop there. First of all, the evolution of the target application can go on almost entirely without disturbing the developed aspects. In this case, new versions of the demo application have been released during this research, and the latest version seems to work with the aspects developed for an earlier version. This again proves two points at least in the circumstances of this case. Firstly, the developers of the target application can be completely oblivious of the work done for the aspect side. Secondly, and resulting from the first one, the aspect-oriented way of adding logging to the target program liberated those developers from having to remember to write a logger call for each button they implemented. Although this might seem easy enough in simple programs, the automated version of the aspect-oriented way should result in higher consistency. In general, the one thing in the way of this oblivious evolution could be the alterations to the source code where the pointcuts have been made. But even in that case, all that would be needed is to update the corresponding pointcuts.

As described, the implementation in this case used clear-box quantification. In relation to black-box quantification it has been evaluated as more difficult for reuse. However, when considering the reuse options for this case, it seems quite straightforward to use the aspects developed for this research in at least other applications, which have been developed with the Vaadin Framework. As the source code of the demo application was not changed during the development of the aspects, the pointcuts where made on standard method calls of the Vaadin Framework and its classes found in every Vaadin Framework application. These, again, are also bound to be used with any application

developed with that framework, and therefore the reuse of the now developed aspects seems to be at least possible with different Vaadin Framework applications.

On the other hand, one should consider also the reuse with other applications than the ones developed with the Vaadin Framework. In this sense, the aspects developed for this research would not work almost at all. First off, one should implement the pointcuts differently as the same method calls and classes would not be available. Secondly, the now developed aspect-oriented collector relies heavily on button click listeners to be available and if those kinds of tools are not to be used, the whole reuse must be redesigned with a different approach. Finally, although the data storing was done in a container, which probably adapts quite easily to a SQL table, it still restricts the straightforward reuse to the applications of the Vaadin Framework having the same container class.

However, if the decision for data storing would have been either sending the data to an external API or logging it to a text file, the implementation would not have to be changed when the aspects are reused. With these approaches though come other types of drawbacks. An external API would have required work amounts of another scale, lucrative only if real reuse options would have existed. Additionally, the use of an external API adds up problems to a different level considering information security, data ownership, and legal issues. Similarly, the use of a text file for logging could be a tempting option when considering only the point of reuse. This approach seems to be contrary to the external API logging though, as the logging itself should be quite simple to implement. This is probably why it seems many programs overall are designed to do at least some logging of this kind. The drawback comes however in the Data Preparation point rather than from the Data Understanding, when considering the Data Mining Method presented above. Although the data logging itself might be easy, there inevitably is more preprocessing in data from a text file than from a SQL table.

6.2.2. NITCAD evaluation: Information needs in software analytics

Considering the information needs of developers in software analytics, one of the most interesting subjects to them was to find out the situations, code, and the states of a system, which lead to failures. Although this research was not focused on software errors, it became quite obvious how intuitive the approach of AOP would be in collecting information on the points of system failures. Although the defining of a pointcut to these points in the execution flow might be simple, the real data collecting could be a lot more difficult in practice as some of the contextual data can be perished in case of a failure. However, even the data collected by the now developed NITCAD should be helpful in understanding at least the former execution flow of the target application and the system-user interaction leading to an error.

From a more managerial point of view, the information needs focused on understanding customers and their behavior. As mentioned, automated usage analysis, which is based on real execution data, can lead to business advantages such as customer satisfaction, customer retention, and finally to increase in sales. As the aspect-oriented collector was able to produce data on sessions and buttons, clickstream data could be formed. Done in a real production environment with actual clients and customers, this could be used for understanding for example which features attract customers the best.

On the other hand, managers did not want analytics to help with the future, but rather with the past and the present. These information needs concentrated on topics such as refactoring, release planning, and judging stability. As NITCAD was able to record data such as timestamps, sessions, and button captions, conclusions can be made on when the users are using the system, what features they are using, and which features lead to stopping the use. These conclusions can then help with the corresponding decisions such as when to make releases and which features need refactoring.

Next, the collected types of data are evaluated by comparing them and their derivatives to the segments of software operation knowledge. First of all, knowledge about the performance includes information such as elapsed time, transaction throughput, and response time. As the aspect-oriented collector was able to log information on the last timestamp of a request, these pieces of information could be used for calculating measures for the performance of the system. With adding the session information to the mix, preliminary measurement of throughput could be formed for example in terms of how long one user session lasts. In addition, the collected information of response time for the last request lands obviously straight to the path of performance measuring.

Secondly, software operation knowledge includes quality attributes such as number of exceptions, crash report details, and mean time between failures. As mentioned, the focus of this research was not on finding out possible pointcuts in failure situations. Based on the literature review on AOP however, it seemed not only possible but also quite simple to attach at least some kind of functionality to those points of execution as well. Additionally, the quality part of software operation knowledge includes quality-in-use, which again includes things such as end-user productivity and end-user satisfaction. Of these, the aspect-oriented collector provides information for measuring end-user productivity in somewhat same manner as was described with the performance measuring. In this case, in addition to the information about the session and timestamps, the button caption or URI fragment information could be added up to form specific tasks for end-users, and by measuring how much time they spend on a task, the end-user productivity could be analyzed.

Thirdly, the usage segment of software operation knowledge turned out to be the most fruitful source of information. This resulted on one hand from the aspect-orientation of the collector and on the other from the abundant context information available in a

Vaadin Framework application. As the usage segment consists of interface paths, method calls, and object initiations, AOP had all the answers to them with its pointcuts being able to be defined to attach specifically to method calls and object initiations. Fortunately, the Vaadin Framework covered the other end by providing information such as time and date of request, requested source, status of the request, and http method. However, there were pieces of information such as size of the returned object, referring source, and user-agent of the client, which are mentioned as parts of the usage knowledge but which could not have been collected in this case.

Finally, end-user feedback could be one of the key information needs and a critical part of software operation knowledge, but in this case it was out of the focus of this research. This segment consists of pieces of information such as end-user appreciation, criticism on certain software usage aspects, and general software experience. Although these types of information were ignored this time, the potential of using AOP to add functionalities for collecting information such as this became apparent. For example, the same pointcuts as in this case could be used, but in the advice code the functionality of inserting an extra feedback query to users would be added.

7. EVALUATION

This chapter evaluates the conducted research. Firstly, contributions of the study are described from both academic and practical standpoints. Secondly, the contributions of this study are compared to related work. Thirdly, the limitations of this study are examined in terms of validity and reliability, which leads finally to the presentation of potential research areas for future topics and extensions.

7.1. Contributions of the study

The objectives of this study were twofold. On one hand, the study was an academic research effort. To evaluate it in this sense, the next subsection reviews the research questions and how the study was able to give answers to them. On the other hand, there were also objectives of practical nature. The contributions of this kind are described in the second subsection.

7.1.1. Academic contributions

Of the three research questions, the first one was set to assist in understanding the principle concepts of AOP (*What kind of advantages AOP offers for collecting software operation data?*). To answer this, a literature review was conducted on AOP and these results were described in terms of how aspects can be technically implemented as well as how the non-invasiveness of the aspect-oriented approach benefits collecting software operation data. Although particularly the answers for the technicalities and nature of AOP were drawn from previous research and no new knowledge was produced, this was a crucial part of the design science method and an important prerequisite considering the rest of the study. In addition, Chapter 3 contributes as a short summary of the characteristics and benefits of AOP as well as an introduction to the paradigm and a language implementing it. The distinguishing characteristics of AOP are related to obliviousness and quantification. These result in the vital advantage for the whole software development in terms of the non-invasiveness, which again leads to easier maintenance, reuse, and evolution of software.

Similar to the way the first research question was answered, the second research question was approached also with reviewing literature. Considering the research process of design science, this literature review assisted in the phase of defining objectives of the solution. The question was formulated as: *What kind of software operation knowledge is needed for software analytics?* Firstly, the common technologies of analytics were studied, eventually focusing specifically on software

analytics. Secondly, information needs in software development were examined from the perspectives of developers and managers. These ranged from understanding the system states to understanding the customers and to assisting in refactoring and release planning. Finally, to concretize the way software analytics uses software operation knowledge, a mining method was described.

Contrary to the deductive approach used with the first two research questions, the third one was examined inductively. Formulated as: *How NITCAD is able to deliver the benefits AOP paradigm promises and to answer to the information need of software analytics?*, the third question obviously merges the answers of the first two to use them in this specific situation. The developed tool was first presented as a software artifact and then evaluated with an informed argument from the two different points of the third research question. The informed argument was based on the former literature reviews.

Considering the first point, NITCAD was clearly able to deliver the key benefits of AOP, such as the non-invasiveness of inserting an additional feature to an existing application. This resulted in easier maintenance and evolution of the original application. Although the benefit of reusing the developed tool seems to exist, it could not be made certain within this research. In view of the second point, the developed tool answered specifically the information needs. It collected crucial data of the operation context and stored these pieces of data to an SQL-type of a container making it information. Thus, if the research question had asked for knowledge or intelligence needs, the evaluation would have had to deny the success of the developed tool. However, now NITCAD collected the necessary information and the turning of this information into knowledge was left out of the scope of this research. As the crucial types of information, such as session identifiers, last request timestamps, and URI fragments, were recorded and also a mining method was presented, it should form a solid groundwork for continuing towards developing a tool that can answer straight also to higher-level needs in software development.

7.1.2. Practical considerations

Of the twofold objectives of this research, the second one considered the practical contributions. In this regard, an increase in the needs of gathering software operation knowledge is anticipated as the decision making is seen to become more and more data-driven also in software development. Therefore, solutions such as this implementation of AOP to the context of software analytics, contribute also to practice by increasing the knowledge about the different technical options available for gathering data.

To further elaborate the practical contribution of this research, the aspect-oriented way of gathering software operation knowledge is compared with other methods as follows. Firstly, considering the manual methods for gathering software operation data, the automatic way of AOP obviously has an advantage in terms of collecting higher

quantities of data in a shorter period of time. On the other hand, the use of these manual methods can still be justified as they can go a lot deeper in the details of a single user session. In addition, they can produce relevant information also with small amounts of users. On the contrary, greater amounts of users might be required by automatic methods such as A/B testing. However, methods such as querying users for their use of a system might turn out to be less trustworthy, as automatic methods collect data without the user being aware of the collecting and the collected data will not be affected by users' opinions.

Secondly, when compared with other automatic methods of collecting software operation data, the aspect-oriented way has some novel characteristics. Application specific data gathering systems might leave source code tangled and lack reuse options, whereas the benefits of AOP should support specifically these attributes. Collection frameworks should be similar to AOP in that sense, and both of them can also produce data that is consistent through the use in different systems. However, collection frameworks require extensive development decisions upfront, while aspects can be inserted non-invasively after the development of a target application as seen in this research. Further on, COTS tools provide the same kind of reusability and consistency but lack the customizability of the aspect-oriented approach. All in all, this research contributes to practice by introducing AOP as a new way of collecting software operation data. This novel approach has advantages, such as the non-invasiveness, when compared with the other methods. However, its potential disadvantages, such as the effects on performance and requirements of additional weaving operations, should be examined more closely in the future.

Finally, the aspect-oriented NITCAD for the Vaadin Framework was developed as an outcome of this research. This tool will be distributed as an open-source project hosted by Github, and its diffusion will be promoted with a blog post in Vaadin's website after the publication of this thesis. As an open-source tool, any developer using the Vaadin Framework can download the tool and apply its data collecting features. At the same time, Vaadin can use NITCAD as a starting point in the development of an extra feature for their Framework.

7.2. Related work

Baysal et al. (2012, p. 98) point out how software repository mining has been focusing on development artifacts such as source code and version control metadata. In order to extend this scope, they have conducted research on mining web logs of usage data. Their previous work suggests that by studying this web usage data, one can infer knowledge on for example user adoption trends. This again can be valuable for purposes such as evaluating the success of a product and performing market analysis. (Baysal et al. 2012, p. 98)

In a wider perspective, the results by Baysal (2014) show that software analytics can inform evidence-based decision making on topics such as user adoption of a software project, code review processes, and improving developers' awareness on their daily tasks and activities. In addition to these, Marciuska (2013) points out how current software products are developed incrementally in general either by adding new features on top of existing ones or by enhancing them. This leads to software companies' need of measuring and monitoring the perceived value of these features. As this value is constantly changing, current methods such as user surveys have to be repeated frequently, which again make them an expensive option. Therefore, more automated methods for identifying and monitoring the usage of features are needed. (Marciuska 2013)

As a solution for this kind of a problem, the use of AOP was studied in this research. However, AOP has been studied also as an approach to testing amongst other things. For example, Metsä (2009) has presented how aspect-orientation can assist in capturing system-wide, cross-cutting concerns and modularizing them into manageable units for testing. Similar to the benefits found in this research, the implementation of testing was done in a non-invasive manner. Thus, the aspect-oriented approach was seen to propose "a technique to implement testware without actually touching the code. (Metsä 2009)"

Although in general the use of AOP has been viewed as beneficial, including cases such as this and the aforementioned approach to testing, there have been contradicting voices as well. For example, Steimann (2006) finds that AOP actually works against the purposes of independent development and understandability of programs. However, in a systematic literature review by Ali et al. (2010) AOP was found to have a positive effect on performance, code size, modularity, and evolution. In their study, which compared AOP with traditional programming languages such as object-oriented and structured programming languages, the effects of AOP on cognition and language mechanism were found to be less likely to be positive.

7.3. Study limitations

Runeson & Höst (2009, p.153-154) have approached limitations of a study with four different categories. These are construct validity, internal validity, external validity, and reliability. Firstly, construct validity reflects how the operational measures actually represent what the researcher has had in mind. In this case, the study involved only the researcher and thus the construct validity can be regarded irrelevant to be examined further. Secondly, internal validity concerns the evaluation of causal relations. This study involved evaluation of the Vaadin Framework and shortly also how NITCAD could be reused with other applications developed with the framework. This evaluation took into account only the fact that a specific method seems to be a common one in Vaadin Framework applications. However, no other factors were analyzed considering their effects on the reuse possibilities. All in all, this study's methods have been

examined by two supervisors, and in that sense they should be carefully selected to produce the types of results that is the intention.

The third category by Runeson & Höst (2009, p. 154) is external validity. In the regard of how extensively the results are generalizable, this study works only as a starting point for using AOP for collecting software operation data. Although the reuse of NITCAD with other Vaadin Framework applications should be quite straightforward, the validation of this assumption needs further studying. Based on the literature review on AOP, this study provides a new look on how this paradigm could be used. In this sense, this study introduces benefits also outside the context of this specific case.

Finally, Runeson & Höst (2009, p.154) describe reliability as a concern of to what extent the results are dependent on the researcher, who conducted the study. To answer this concern, this study's outcomes have been carefully described firstly in the two literature reviews and secondly with open sourcing NITCAD. Therefore, another researcher should be able to repeat the study and its results, although especially the informed argument method could produce also additional outcomes as in this case the method was carried out solely by one researcher. Additionally, data triangulation was taken into account as same software operation data was collected in two different occasions from two different versions of the demo application.

7.4. Future research topics

This study provides only the starting point in evaluating AOP for gathering software operation data. Although the use of AOP in this area seems at least feasible, one of the key points for future is going to be to find out also the disadvantages of the approach. These include the evaluation of effects of AOP on application performance as well as the possible difficulties in mapping the collected software operation data to specific features of an application. The latter of these might provide extensive advantages for software development and its decision making if it can be done successfully. However, the automated way of mapping software operation data to development process objects such as version control system commits or issue management system tickets, probably requires either complex data mining processes or systematic planning for implementation. Partly for these troublesome issues and partly for the substantial advantages, this kind of mapping provides an interesting future work topic.

In the scope of studying the advantage of reuse of aspect-oriented approach to collecting software operation data, one of the first extensions could be to try out the now developed NITCAD in other Vaadin Framework applications. From there on, using the collecting tool in other Java applications would be a natural follow-up. When a more generalizable view of the advantages and disadvantages of the aspect-oriented approach has been formed, an extensive evaluation between that and other possible methods of

collecting software operation data could be performed. The methods described in this thesis should provide a basis for evaluation of these other possible methods.

Finally, the use of aspect-oriented way to collect software operation data in actual production environments should be studied. A good start for this might be for example in a codecamp environment, but clearly the study should be continued towards a more industrial setting. Along the way, the required layout for the underlying database model could be studied further to find the most suitable way of utilizing the collected data in software development and its decision making.

8. CONCLUSION

In this thesis, the use of Aspect-Oriented Programming was studied for gathering software operation data. Design science was used as the research strategy, and as its outcome a tool for collecting software operation data was developed in AspectJ language. For evaluation, this NITCAD was implemented to collect data from a demo application of the Vaadin Framework. The main research question was formed as:

- *How can AOP be used for supporting software analytics?*

This main question was divided into three sub-questions:

1. *What kind of advantages AOP offers for collecting software operation data?*
2. *What kind of software operation knowledge is needed for software analytics?*
3. *How the developed NITCAD is able*
   3.1. to deliver the benefits AOP paradigm promises and
   3.2. to answer to the information needs of software analytics?

The study was able to find answers to each of the research questions and in that sense it can be seen as a successful research effort. This is especially true for the design science's pursuit of "utility" as the research obviously produced a concrete tool to collect software operation data. However, considering the kinds of data that should be collected, the research presents only one subjective "truth". In this case, the answer was based primarily on one mining method and on one referenced questionnaire study. Thus, the participants of that study influenced greatly also the answers of this study. Although there is no one right answer to the question of what kind of software operation knowledge is needed for software analytics, the direct asking of that question from the developers or managers is likely to produce an answer from just one point of view.

This consideration is important especially with a novel topic such as analytics, as the answer is likely to change quite rapidly alongside the rapid development of relating technology. Therefore, the views and answers described in this study should be regarded definitive by no means, but additional research with different methods and approaches is surely required continuously. However, some initial conclusions can obviously be drawn also from the humble premises of this current study, and they are presented next as follows.

Firstly to answer the first sub-question, literature review of the particular field was conducted. The study found that the advantages of AOP concentrate heavily on the principle of separation of concerns, which again can be seen in the non-invasiveness of

adding extra features to systems. This seems to result in extensive benefits for software development as a whole, as for example the comprehensibility, customizability, and reuse of the systems should be easier than before. Additionally, AOP allows easier system evolution, because it does not require cooperation between the programmers of different parts of a system. Each of these benefits greatly the collecting of software operation data as that kind of features can be added more easily than before and without compromising the evolution of the original system.

Secondly, the next sub-question was approached in a similar manner with a literature review. Overall, software analytics should improve the decision-making in software development by making it more data-driven. The particular topics, which software analytics can help with, include finding out when a system is ready to be released, deciding whether a part of a system should be re-factored or re-written, and understanding customers' behavior. Software operation knowledge can be of high value for software analytics in these pursuits. Collecting software operation knowledge such as clickstream paths can be used for example to analyze which features need re-factoring.

For the third and final sub-question, an aspect-oriented tool for collecting software operation data was developed. The aspect-orientation allowed NITCAD to be developed in separation from the demo application. This way, no alterations to the demo application's source code were needed, which again allowed its evolution without compromising its compatibility with the collecting tool. This non-invasiveness was the most vital benefit, which the AOP paradigm promised, and NITCAD was able to deliver it. This benefit of non-invasiveness might turn out to be quite crucial in cases where the original source code is already written or out of control in some other way. Overall, it should make the maintenance and development of both the aspect and the component programs easier than before. This point was supported already in this study by the evolution of the component program, which was made unaware of the developed aspects – the component program still being compatible with the aspects and not requiring any additional alterations to either program.

The Vaadin Framework provided NITCAD with abundant contextual data, which can be used for evaluating user behaviors. Considering the information needs of software analytics, NITCAD was able to answer them with providing data for example for creating clickstream information. As manual methods might prove to be too costly to be used for collecting such data, AOP provides a new way of collecting software operation data without compromising productivity. NITCAD can be reused with other Vaadin Framework applications and it provides Vaadin a starting point for creating an additional analytics feature for their framework. Academically, this thesis presents the novel context of software analytics for taking advantage of AOP's primal and solid utilization in logging.

BIBLIOGRAPHY

Ali, M. S., Ali Babar, M., Chen, L., Stol, K. J. 2010. A systematic review of comparative evidence of aspect-oriented programming. Information and software Technology, 52, 9, 871-887.

Asif, M., Reddy, Y. R. 2013. JIFFY: A framework for encompassing aspects in testing and debugging software. In Automation of Software Test (AST), 2013 8th International Workshop on, pp. 146-149. IEEE.

Baskerville, R., Meyers, M.D. 2004. Special Issue on Action Research in Information Systems: Making IS Research Relevant to Practice – Forward. MIS Quarterly, 28, 3, pp 329-335.

Baysal, O. 2014. Supporting Development Decisions with Software Analytics. Doctoral thesis. Waterloo, Ontario, Canada, University of Waterloo. 197 pages.

Baysal, O. 2013. Informing Development Decisions: From Data to Information. ICSE 2013 Doctoral Symposium, San Francisco, California, United States of America, pp. 1407-1410.

Baysal, O., Holmes, R., Godfrey, M. W. 2012. Mining usage data and development artifacts. In Mining Software Repositories (MSR), 2012 9th IEEE Working Conference on, pp. 98-107. IEEE.

Begel, A., Zimmermann, T. 2014. Analyze This! 145 Questions for Data Scientists in Software Engineering. ICSE 2014, May 31 – June 7, 2014, Hyderabad, India. 12 pages.

Buse, R., Zimmermann, T., Information needs for software development analytics, Microsoft Research, Tech. Rep. MSR-TR-2011-8, 2011, 16 pages. Available: http://131.107.65.14/pubs/144543/MSR-TR-2011-8.pdf

Chaudhary, R., Chatterjee, R. 2013. Essence of reusability in aspect-oriented systems. ACM SIGSOFT Software Engineering Notes, 38, 3, pp. 1-5.

Chen, H., Chiang, R. H., Storey, V. C. 2012. Business Intelligence and Analytics: From Big Data to Big Impact. MIS quarterly, 36, 4, pp. 1165-1188.

Choo, C. W. 1996. The knowing organization: how organizations use information to construct meaning, create knowledge and make decisions. International journal of information management, 16, 5, pp. 329-340.

Clow, D. 2012. The learning analytics cycle: closing the loop effectively. In Proceedings of the 2nd International Conference on Learning Analytics and Knowledge, pp. 134-138. ACM.

Cole, R., Purao, S., Rossi, M., Sein, M. 2005. Being proactive: where action research meets design research. ICIS 2005 Proceedings, paper 27.

Davenport, T.H., Harris, J.G., Morison, R. 2010. Analytics at work: Smarter Decisions, Better Results. Boston, Massachusetts, United States of America, Harvard Business School Publishing. 214 pages.

Davenport, T.H., Prusak, L. 1998. Working knowledge: how organizations manage what they know. Harvard Business School Press, 199 pages.

Elrad, T., Filman, R. E. 2001. Aspect-oriented programming. Communications of the ACM. 44, 10, pp. 29-32.

El-Ramly, M., Stroulia, E. 2004. Mining software usage data. In Proceedings of 1st International Workshop on Mining Software Repositories (MSR'04), pp. 64-8.

Filman, R. E., Elrad, T., Clarke, S., Aksit, M. 2004) Aspect-oriented software development. Boston, MA, USA, Pearson Educations, Inc. 755 pages.

Filman, R. E., Friedman, D. P. 2000. Aspect-oriented programming is quantification and obliviousness. In Workshop on Advanced separation of Concerns, OOPSLA.

Fowler, M., Highsmith, J. 2001. The agile manifesto. Software Development, 9, 8, pp. 28-35.

Fuggetta, A., Di Nitto, E. 2014. Software process. In Proceedings of the FOSE'14, Future of Software Engineering, May 31 – June 7, 2014, Hyderabad, India. pp. 1-12. ACM.

Germann, F., Lilien, G. L., Fiedler, L., Kraus, M. 2014. Do Retailers Benefit from Deploying Customer Analytics?. Journal of Retailing, 90, 4, pp. 587-593.

Google Inc. 2014. Google Analytics: Yritystason verkkoanalyysi. http://www.google.com/analytics/. 15.9.2014.

Great Place to Work® Institute, Inc. 2014. Parempi työelämä: Vaadin Oy. http://www.parempityoelama.fi/parhaat-tyopaikat/yleinen-sarja/vaadin/. 15.9.2014

Greller, W., Drachsler, H. 2012. Translating Learning into Numbers: A Generic Framework for Learning Analytics. Educational Technology & Society, 15, 3, pp. 42–57.

Grönroos, M. 2013. Book of Vaadin: Vaadin 7 Edition – 1st Revision. Turku, Finland, Vaadin Ltd. 681 pages.

Halkidi, M., Spinellis, D., Tsatsaronis, G., Vazirgiannis, M. 2011. Data mining in software engineering. Intelligent Data Analysis, 15, 3, pp. 413-441.

Hassan, A. E., Xie, T. 2010. Software intelligence: the future of mining software engineering data. In Proceedings of the FSE/SDP workshop on Future of software engineering research, pp. 161-166. ACM.

Hertzum, M., Jacobsen, N. E. 2001. The evaluator effect: A chilling fact about usability evaluation methods. International Journal of Human-Computer Interaction, 13, 4, pp. 421-443.

Hevner, A. R., March, S. T., Park, J., Ram, S. 2004. Design science in information systems research. MIS quarterly. 28, 1, pp. 75-105.

Hirsjärvi, S., Remes, P., Sajavaara, P. 2007. Tutki ja kirjoita. 13th edition, Finland, Tammi. 448 pages.

Holzinger, A. 2005. Usability engineering methods for software developers. Communications of the ACM, 48, 1, pp. 71-74.

Huomo, T., Järvinen, J., Kettunen, P., Kuvaja, P., Koivisto, A., Lassenius, C., Lehtovuori, P., Lilja, S., Miettinen, S., Mikkonen, T., Münch, J., Männistö, T., Oivo, M., Partanen, J., Porres, I., Still, J., Tyrväinen, P. 22.4.2013. Strategic Research Agenda. http://www.n4s.fi/wordpress/wp-content/uploads/2014/03/SRA-Need4Speed-4-2.pdf. 22.12.2014.

Johnson, P. 2013. Searching under the streetlight for useful software analytics. IEEE Software. 30 ,4, pp. 57-63.

Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C., Loingtier, J. M., Irwin, J. 1997. Aspect-oriented programming. In Proceedings of ECOOP'97, 11th European Conference, Jyväskylä, Finland, June 9-13, 1997. Lecture Notes in Computer Science, pp. 220-242. Springer.

Kiczales, G., Hilsdale, E., Hugunin, J., Kersten, M., Palm, J., Griswold, W. G. 2001a. An overview of AspectJ. In Proceedings of ECOOP 2001—Object-Oriented Programming, 15th European Conference Budapest, Hungary, June 18–22, 2001. Lecture Notes in Computer Science Volume, pp. 327-354. Springer Berlin Heidelberg.

Kiczales, G., Hilsdale, E., Hugunin, J., Kersten, M., Palm, J., Griswold, W. G. 2001b. Getting started with AspectJ. Communications of the ACM, 44, 10, pp. 59-65.

Ko, A. J., DeLine, R., Venolia, G. 2007. Information needs in collocated software development teams. In Proceedings of the 29th international conference on Software Engineering, pp. 344-353. IEEE Computer Society.

Kohavi, R., Longbotham, R., Sommerfield, D., Henne, R. M. 2009. Controlled experiments on the web: survey and practical guide. Data Mining and Knowledge Discovery, 18, 1, pp. 140-181.

Koivuviita, J. 24.01.2013. It's demo time. https://vaadin.com/blog/-/blogs/it-s-demo-time. 23.9.2014.

Kristjansson, B., Van der Schuur, H. 2009. A survey of tools for software operation knowledge acquisition. Technical report UU-CS-2009-028. Department of Information and Computing Sciences, Utrecht University.

Kumar, N., Sosale, D., Konuganti, S. N., Rathi, A. 2009. Enabling the adoption of aspects-testing aspects: a risk model, fault model and patterns. In Proceedings of the 8th ACM international conference on Aspect-oriented software development pp. 197-206. ACM.

Larman, C. 2003. Agile and Iterative Development: A Manager's Guide. Addison-Wesley Professional, 368 pages.

Lieberherr, K., Orleans, D., Ovlinger, J. 2001. Aspect-Oriented Programming with adaptive methods. Communications of the ACM, 44, 10, pp. 39-41.

Liebowitz, L. 2006. Strategic intelligence, Business intelligence, Competitive intelligence and knowledge management. Auerbach publications, New York, United Stated of America, 223 pages.

Loshin, D. 2012. Business Intelligence: The Savvy Manager's guide. Second edition, Morgan Kauffmann Publishers, San Francisco, United States of America, 370 s.

Maple, S. 13.6.2013. The Curious Coder's Guide to Java Web Frameworks: Vaadin. http://zeroturnaround.com/rebellabs/the-curious-coders-guide-to-java-web-frameworks-vaadin. 15.9.2014

March, S.,T., Smith, G.,F. 1995. Design and natural science research on information technology. Decision Support Systems, 15, 4, pp. 251-266.

Marciuska, S. 2013. Towards development of field methods and tools for feature value monitoring. Doctoral thesis. Bolzano, Italy, Free University of Bozen-Bolzano. 126 pages.

Menzies, T., Bird, C., Zimmermann, T. 2014. Analyzing software data: after the gold rush (a goldfish-bowl panel). In Companion Proceedings of the 36th International Conference on Software Engineering, pp. 103-104. ACM.

Metsä, J. 2009. Aspect-Oriented Approach to Testing – Experiences from a Smartphone Product Family. Doctoral thesis. Tampere, Finland, Tampere University of Technology. 82 pages.

Nakatani, K., Chuang, T. T., 2011. A web analytics tool selection method: an analytical hierarchy process approach. Internet Research, 21, 2, pp. 171-186.

Olkkonen, T. 1994. Johdatus teollisuustalouden tutkimustyöhön. 2nd edition. Teknillinen korkeakoulu, Teollisuustalouden laitos, Finland. 143 pages.

Pachidi, S., Spruit, M., Van De Weerd, I. 2014. Understanding users' behavior with software operation data mining. Computers in Human Behavior, 30, pp. 583-594.

Peffers, K., Tuunanen, T., Rothenberger, M. A., Chatterjee, S. 2007. A design science research methodology for information systems research. Journal of management information systems. 24, 3, pp. 45-77.

Popovici, A., Gross, T., Alonso, G. 2002. Dynamic weaving for aspect-oriented programming. In Proceedings of the 1st international conference on Aspect-oriented software development (AOSD '02). ACM, New York, NY, USA, p.141-147.

Pöntelin, T. 17.5.2011. Vaadin Scalability Study – QuickTickets. https://vaadin.com/blog/-/blogs/vaadin-scalability-study-quicktickets. 23.9.2014.

Ries, E. 2011. The Lean Startup. First edition, United States, Crown Business, 336 pages.

Runeson, P., Höst, M. 2009. Guidelines for conducting and reporting case study research in software engineering. Empirical software engineering, 14, 2. pp. 131-164.

Sadeoja, T., 2014. Vaadin Oy Taloustiedot. http://yritys.taloussanomat.fi/y/vaadin-oy/helsinki/1613563-9/. 15.9.2014

Saunders, M., Lewis, P., Thornhill, A. 2009. Research methods for business students. Fifth edition, England, Pearson Education Limited. 614 pages.

Sein, M., Henfridsson, O., Purao, S., Rossi, M., Lindgren, R. 2011. Action design research. MIS Quarterly. 35, 1, pp. 37-56.

Sharma, R., Reynolds, P., Scheepers, R., Seddon, P. B., Shanks, G. G. 2010. Business Analytics and Competitive Advantage: A Review and a Research Agenda. DSS, pp. 187-198.

Simon, P. 2013. Too Big to Ignore: The Business Case for Big Data. Hoboken, New Jersey, United States of America, John Wiley & Sons, Inc. 232 pages.

Steimann, F. 2006. The paradoxical success of aspect-oriented programming. In ACM Sigplan Notices, 41, 10, pp. 481-497. ACM.

Subotic, S., Bishop, J. 2005. Emergent behaviour of aspects in high performance and distributed computing. In Proceedings of the 2005 annual research conference of the South African institute of computer scientists and information technologists on IT research in developing countries, pp. 11-19. South African Institute for Computer Scientists and Information Technologists.

Thierauf, R. 2001. Effective Business Intelligence Systems. Greenwood Publishing Groups, Westport, CT, United States of America, 370 pages.

van der Schuur, H., Jansen, S., Brinkkemper, S. 2010. A Reference Framework for Utilization of Software Operation Knowledge. In SEAA'10: 36th EUROMICRO Conference on Software Engineering and Advanced Applications. pp. 245–254. IEEE

van der Schuur, H., Jansen, S., Brinkkemper, S. 2011. The power of propagation: on the role of software operation knowledge within software ecosystems. In Proceedings of the International Conference on Management of Emergent Digital EcoSystems (MEDES '11). pp. 76-84. ACM.

Watson, H. J., Wixom, B. H. 2007. The current state of business intelligence. Computer, 40, 9, pp. 96-99.

Wiggins, A. 2007. Information architecture: Data-driven design: Using web analytics to validate heuristics system. Bulletin of the American Society for Information Science and Technology, 33, 5, pp. 20-24.

Williams, L., Cockburn, A. 2003. Agile software development: it's about feedback and change. IEEE Computer, 36, 6, pp. 39–43.

Winter, R. 2008. Design science research in Europe. European Journal of Information Systems, 17, 5, pp. 470-475.

Yritys- ja yhteisötietojärjestelmä, 2014. YTJ-tietopalvelu – Yrityshaku: Vaadin Oy. https://www.ytj.fi/yritystiedot.aspx?yavain=1385965&kielikoodi=1&tarkiste=63A513CDCDC2ADD75040F93F2A577DD689F11BD1&path=1547;1631;1678. 15.9.2014.

Zhang, D., Dang, Y., Lou, J. G., Han, S., Zhang, H., Xie, T. 2011. Software analytics as a learning case in practice: Approaches and experiences. In Proceedings of the International Workshop on Machine Learning Technologies in Software Engineering, pp. 55-58. ACM.

APPENDIX 1: Source code of DataLogger.java

```
package com.example.analytics_dashboard;
import java.util.Date;
import com.vaadin.data.util.IndexedContainer;
import com.vaadin.server.Page;
import com.vaadin.server.VaadinService;
import com.vaadin.server.VaadinSession;
import com.vaadin.ui.Button;
import com.vaadin.ui.Component;
import com.vaadin.ui.HasComponents;
import com.vaadin.ui.Table;
import com.vaadin.ui.UI;
import com.vaadin.ui.Button.*;

public class DataLogger {

	private IndexedContainer clicks;

	DataLogger(){
		clicks = new IndexedContainer();

		clicks.addContainerProperty("Session ID", String.class, "no_value");
		clicks.addContainerProperty("LastRequestDuration", Long.class, null);
		clicks.addContainerProperty("LastRequestTimestamp", Long.class, null);
		clicks.addContainerProperty("SessionState", String.class, "no_value");

		clicks.addContainerProperty("Method", String.class, "no_value");
		clicks.addContainerProperty("Uri Fragment", String.class, "no_value");

		clicks.addContainerProperty("Page", String.class, "no_value");

		clicks.addContainerProperty("UI EmbedId", String.class, "no_value");
		clicks.addContainerProperty("UI Id", String.class, "no_value");
		clicks.addContainerProperty("UI ID", Integer.class, null);

		clicks.addContainerProperty("Button Connector ID", String.class,
"no_caption");

		clicks.addContainerProperty("Parent Class", String.class, "no_value");
		clicks.addContainerProperty("Parent ID", String.class, "no_value");

		clicks.addContainerProperty("Button Caption", String.class, "no_value");
		clicks.addContainerProperty("Button ID", String.class, "no_value");
		clicks.addContainerProperty("Click X", Integer.class, null);
		clicks.addContainerProperty("Click Y", Integer.class, null);

		clicks.addContainerProperty("Timestamp", java.util.Date.class, null);
	}

	public void logButtonClick(Button b, ClickEvent event){
		Object itemID = clicks.addItem();
		Date d = new Date();

		clicks.getContainerProperty(itemID, "Session
ID").setValue(UI.getCurrent().getSession().getSession().getId());
```

```
                clicks.getContainerProperty(itemID,
"LastRequestDuration").setValue(VaadinSession.getCurrent().getLastRequestDuration());
                clicks.getContainerProperty(itemID,
"LastRequestTimestamp").setValue(VaadinSession.getCurrent().getLastRequestTimestamp());
                clicks.getContainerProperty(itemID,
"SessionState").setValue(VaadinSession.getCurrent().getState().toString());

                clicks.getContainerProperty(itemID,
"Method").setValue(VaadinService.getCurrent().getCurrentRequest().getMethod());
                clicks.getContainerProperty(itemID, "Uri
Fragment").setValue(Page.getCurrent().getUriFragment());

                clicks.getContainerProperty(itemID,
"Page").setValue(Page.getCurrent().toString());

                clicks.getContainerProperty(itemID, "UI
EmbedId").setValue(UI.getCurrent().getEmbedId());
                clicks.getContainerProperty(itemID, "UI
Id").setValue(UI.getCurrent().getId());
                clicks.getContainerProperty(itemID, "UI
ID").setValue(UI.getCurrent().getUIId());

                clicks.getContainerProperty(itemID, "Button Connector
ID").setValue(b.getConnectorId());

                clicks.getContainerProperty(itemID, "Parent
Class").setValue(b.getParent().getClass().toString());
                clicks.getContainerProperty(itemID, "Parent
ID").setValue(b.getParent().getId());

                clicks.getContainerProperty(itemID, "Button
Caption").setValue(b.getCaption());
                clicks.getContainerProperty(itemID, "Button ID").setValue(b.getId());
                clicks.getContainerProperty(itemID, "Click
X").setValue(event.getClientX());
                clicks.getContainerProperty(itemID, "Click
Y").setValue(event.getClientY());

                clicks.getContainerProperty(itemID, "Timestamp").setValue( d );
        }

        public void setTable(Table table){
                table.setContainerDataSource(clicks);
        }

        public void printButtonDetails( Component c ){
                System.out.println("b: " + c);
                System.out.println("Button ID: " + c.getId());
                System.out.println("Caption: " + c.getCaption());
                System.out.println("Class: " + c.getClass());
                System.out.println("UI: " + c.getUI());
                System.out.println("Parent: " + c.getParent());
        }

        public void printEventDetails( ClickEvent event ){
                System.out.println("Event: " + event);
                System.out.println("ClientX: " + event.getClientX());
                System.out.println("ClientY: " + event.getClientY());
                System.out.println("Class: " + event.getClass());
                System.out.println("RelativeX: " + event.getRelativeX());
                System.out.println("RelativeY: " + event.getRelativeY());
                System.out.println("toString: " + event.toString());
                System.out.println("Button: " + event.getButton());
                System.out.println("Component: " + event.getComponent());
```

```
                System.out.println("Connector: " + event.getConnector());
                System.out.println("Source: " + event.getSource());
        }

        public void printLayoutDetails( HasComponents layout ){
                System.out.println("Layout: " + layout);
                System.out.println("Caption: " + layout.getCaption());
                System.out.println("Description: " + layout.getDescription());
                System.out.println("ID: " + layout.getId());
                System.out.println("Class: " + layout.getClass().getName());
                System.out.println("Parent: " + layout.getParent());
                System.out.println("toString: " + layout.toString());
                System.out.println("UI: " + layout.getUI());
        }

        public void printUIDetails(){
                System.out.println("------UI-DETAILS---------------");
                System.out.println("Current UI: " + UI.getCurrent());
                System.out.println("Caption: " + UI.getCurrent().getCaption());
                System.out.println("Component count: " +
                                UI.getCurrent().getComponentCount());
                System.out.println("ConnectorID: " +
                                UI.getCurrent().getConnectorId());
                System.out.println("Description: " +
                                UI.getCurrent().getDescription());
                System.out.println("ID: " + UI.getCurrent().getId());
                System.out.println("Last heart beat timestamp: " +
                                UI.getCurrent().getLastHeartbeatTimestamp());
                System.out.println("Primary style name: " +
                                UI.getCurrent().getPrimaryStyleName());
                System.out.println("Theme: " + UI.getCurrent().getTheme());
                System.out.println("UIId: " + UI.getCurrent().getUIId());
                System.out.println("Class: " + UI.getCurrent().getClass());
                System.out.println("Session: " + UI.getCurrent().getSession());
        }

        public void printPageDetails(){
                System.out.println("------PAGE-DETAILS---------------");
                System.out.println("Current Page: " + Page.getCurrent());
                System.out.println("Location: " + Page.getCurrent().getLocation());
                System.out.println("Web browser: " +
                                Page.getCurrent().getWebBrowser());
                System.out.println("toString: " +
                                Page.getCurrent().toString());
                System.out.println("Uri fragment: " +
                                Page.getCurrent().getUriFragment());
        }
}
```

APPENDIX 2: A table presenting some collected information from the demo application.

Clicks

Session ID	LastRequestDuration	LastRequestTimestamp	Session	Method	Uri Fragment	Page	UI Embedid	UI	UI ID	Button Connector ID	Parent Class	Button Cap	Button ID	Click X	Click Y	Timestamp
508ED15746E3B4F4595D01D70FB18378	38	1,415,874,507,627	OPEN	POST		com.vi	analyticsda		1	29	class com.va	Sign In		1,111	481	Nov 13, 2014 1
508ED15746E3B4F4595D01D70FB18378	110	1,415,874,536,916	OPEN	POST	l/tables	com.vi	analyticsda		1	47	class com.va	Tables		66	436	Nov 13, 2014 1
508ED15746E3B4F4595D01D70FB18378	86	1,415,874,540,307	OPEN	POST	l/tables	com.vi	analyticsda		1	64	class com.va	Notificatior		1,783	26	Nov 13, 2014 1
508ED15746E3B4F4595D01D70FB18378	3	1,415,874,546,728	OPEN	POST	l/dashboard	com.vi	analyticsda		1	41	class com.va	Dashboard		59	99	Nov 13, 2014 1
508ED15746E3B4F4595D01D70FB18378	125	1,415,874,549,050	OPEN	POST	l/dashboard	com.vi	analyticsda		1	77	class com.va			1,830	15	Nov 13, 2014 1
508ED15746E3B4F4595D01D70FB18378	83	1,415,874,551,025	OPEN	POST	l/dashboard	com.vi	analyticsda		1	92	class com.ex	Save		1,031	482	Nov 13, 2014 1